BroadMX Architecture

Key architectural features for communications performance

MicroUnity

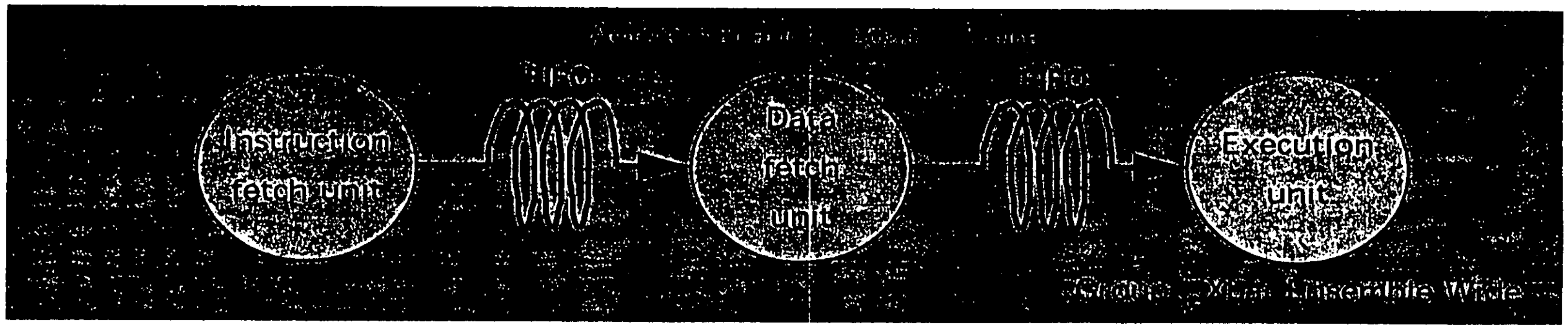
Major Operation Codes

31 24	23 0
major	other
8	24

	Front End: Access				Back End: Execution			
	Address	Branch	Load	Store	Group	Xbar	Ensemble	Wide
MAJOR	0	32	64	96	128	160	192	224
0	[illegible]	BEF16	L16L	S16L	[illegible]	XDEPOSIT	EMULXI	WMULMATXIL
1	[illegible]	[illegible]	L16B	S16B	GADDI		EMULXIU	WMULMATXIB
2	AADDI	[illegible]	L16AL	S16AL	GADDIO		EMULXIM	WMULMATXIUL
3	AADDIUO	[illegible]	L16AB	S16AB	GADDIUO		EMULXIC	WMULMATXIUB
4	[illegible]	[illegible]	L32L	S32L	[illegible]	XDEPOSITU	EMULADDXI	WMULMATXIML
5	ASUBI	[illegible]	L32B	S32B	GSUBI		EMULADDXIU	WMULMATXIMB
6	ASUBIO	[illegible]	L32AL	S32AL	GSUBIO		EMULADDXIM	WMULMATXICL
7	ASUBIUO	[illegible]	L32AB	S32AB	GSUBIUO		EMULADDXIC	WMULMATXICB
8	[illegible]	[illegible]	L64L	S64L	GSETE	XWITHDRAW	ECONXIL	[illegible]
9	ASETNE	[illegible]	L64B	S64B	GSETNE		ECONXIB	[illegible]
10	ASETANDE	[illegible]	L64AL	S64AL	GSETANDE		ECONXIUL	[illegible]
11	ASETANDNE	[illegible]	L64AB	S64AB	GSETANDNE		ECONXIUB	[illegible]
12	ASETL	[illegible]	L128L	S128L	GSETL	XWITHDRAWU	ECONXIML	[illegible]
13	ASETGE	[illegible]	L128B	S128B	GSETGE		ECONXIMB	[illegible]
14	ASETLU	[illegible]	L128AL	S128AL	GSETLU		ECONXICL	[illegible]
15	ASETGEU	[illegible]	L128AB	S128AB	GSETGEU		ECONXICB	[illegible]
16	AANDI	BE	[illegible]	SAS64AL	GANDI	XDEPOSITM	ESCALADDF16	WMULMATXL
17	ANANDI	BNE	LU16B	SAS64AB	GNANDI		ESCALADDF32	WMULMATXB
18	AORI	BANDE	LU16AL	SCS64AL	GORI		ESCALADDF64	WMULMATGL
19	ANORI	BANDNE	LU16AB	SCS64AB	GNORI		ESCALADDX	WMULMATGB
20	AXORI	[illegible]	LU32L	SMS64AL	GXORI	XSWIZZLE	EMULGF8	[illegible]
21	AMUX	BGE	LU32B	SMS64AB	GMUX		EMULG64	[illegible]
22	[illegible]	[illegible]	LU32AL	SMUX64AL	GBOOLEAN		EMULX	[illegible]
23	[illegible]	[illegible]	LU32AB	SMUX64AB	[illegible]		EEXTRACT	[illegible]
24	ACOPYI	[illegible]	LU64L	[illegible]	GCOPYI	XEXTRACT	EEXTRACTI	[illegible]
25	[illegible]	BNV32	LU64B	[illegible]	[illegible]	XSHUFFLE	EEXTRACTIU	[illegible]
26	[illegible]	[illegible]	LU64AL	[illegible]	[illegible]	[illegible]	[illegible]	WTABLEL
27	[illegible]	[illegible]	LU64AB	[illegible]	G8	[illegible]	E8	WTABLEB
28	[illegible]	[illegible]	L8	S8	G16	[illegible]	E16	WSWITCHL
29	[illegible]	BLINKI	LU8	[illegible]	G32	XSHIFTI	E32	WSWITCHB
30	[illegible]	BHINT	[illegible]	[illegible]	G64	XSHIFT	E64	WMINORL
31	AMINOR	BMINOR	LMINOR	SMINOR	G128		E128	WMINORB

SuperSpring

- Decouples Access from Execution



MicroUnity

SuperThread

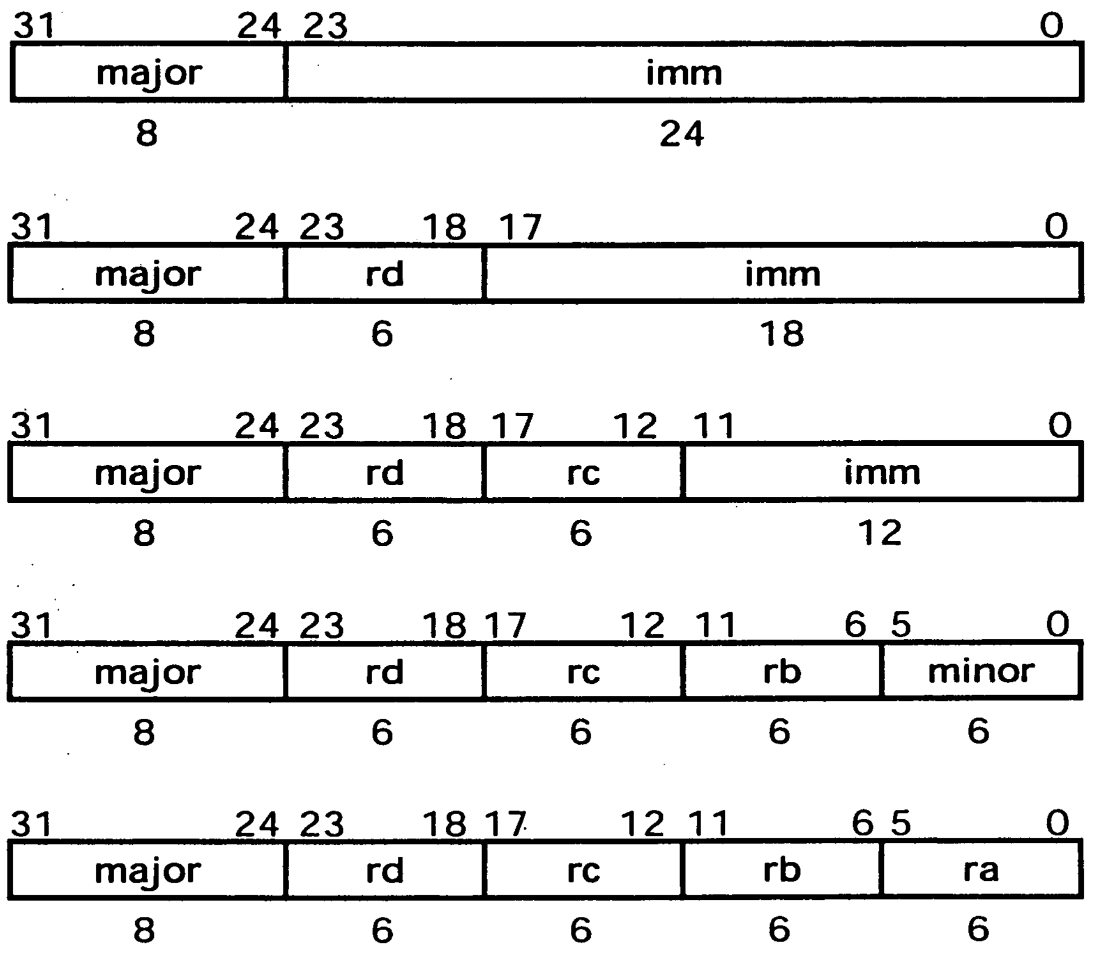
- ■ Simultaneous Multithreading
- ■ Expensive resources ($, X, E, T) shared among threads
  - ◆ improves utilization of resources
- ■ Cheap resources (A, B, L, S) dedicated per thread
  - ◆ keeps branch latency low
  - ◆ enables multiple front-end architectures

MicroUnity

SuperWide

- Memory operand in read-only cache
- Full width register operands
- Full width register result
- Peak utilization of data path bandwidth and flexibility

Instruction Formats



MicroUnity

Address Instructions

- Fixed-point operations over 64-bit addresses
- Add, Subtract, Set-conditional
- Boolean: 2-operand, MUX
- Shift immediate
- Shift left immediate add
- Compare

MicroUnity

Load, Store, Sync Instructions

- Attributes
  - type: signed, Unsigned
  - size: 8, 16, 32, 64, 128
  - alignment: Aligned, unaligned
  - ordering: Little-endian, Big-endian
- Synchronization: 64 A
  - add-, compare-, mux-swap; mux
- Addressing forms
  - register + shifted immediate
  - register + shifted register

Synchronization

- Aligned octlet operations
  - Add-Swap
    - load mem->reg, add reg+mem->mem
  - Compare-Swap
    - load mem->reg, compare reg<->reg,
      if equal, store reg->mem
  - Mux-Swap
    - load mem->reg, mux:mask,reg,mem->mem
  - Mux
    - load mem, mux:mask,reg,mem->mem

MicroUnity

Branch Instructions

■ B.LINK, B.LINK.I	Procedure call
■ B.I	Unconditional
■ B	Procedure return, switch
■ B.DOWN	Gateway return
■ B.BACK	Exception return
■ B.HALT	Interrupt wait
■ B.BARRIER	Instruction-fetch wait
■ Branch conditional	
■ Branch hint	
■ Branch gateway	

MicroUnity

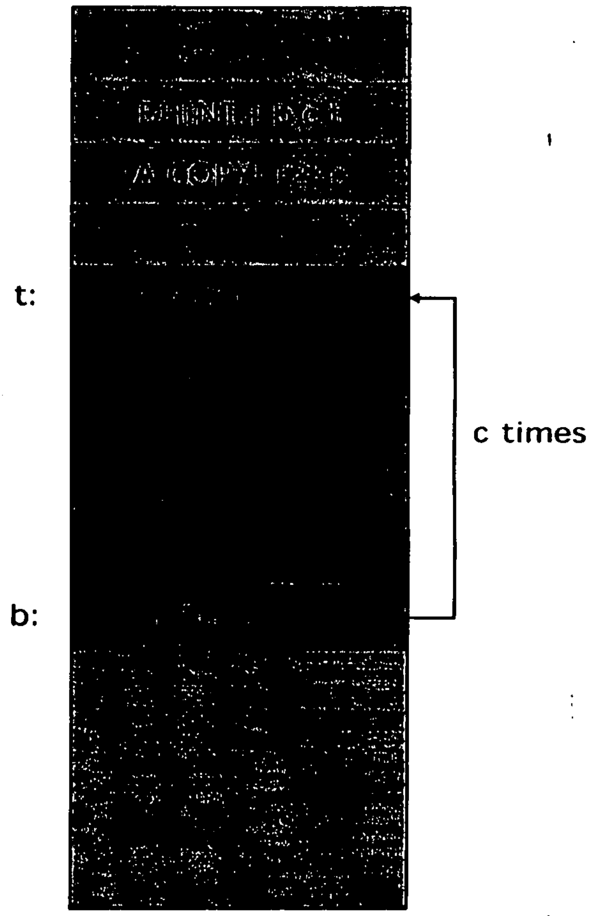
Branch Conditional

- Floating-point: F16 F32 F64 F128
  - B.E.F, B.LG.F, B.L.F, B.GE.F
- Homogeneous Coordinates: 4xF32
  - B.V.F, B.NV.F, B.I.F, B.NI.F
  - Visible: line within viewing cube
  - Invisible: line outside viewing cube
- Fixed-point: 128 bits
  - B.E, B.NE, B.L, B.GE, B.L.U, B.GE.U
  - B.AND.E.Z, B.AND.NE.Z
  - B.E.Z, B.NE.Z, B.L.Z, B.G.Z, B.LE.Z, B.GE.Z

Branch Hint

- Hints for loops, switches, methods
- Fully interruptible

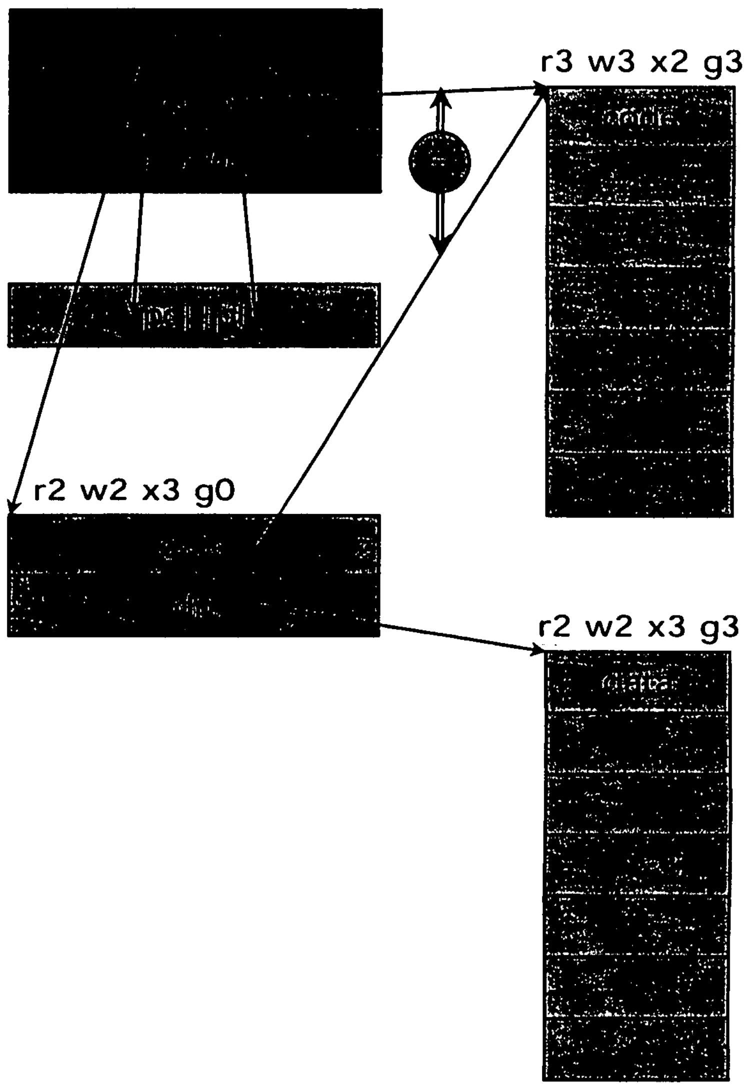
- B.HINT.I b,c,t
- B.HINT b,c,rd
  - Branch at **b** is likely **c** times, to **t/rd**, then is not likely.



Branch Gateway

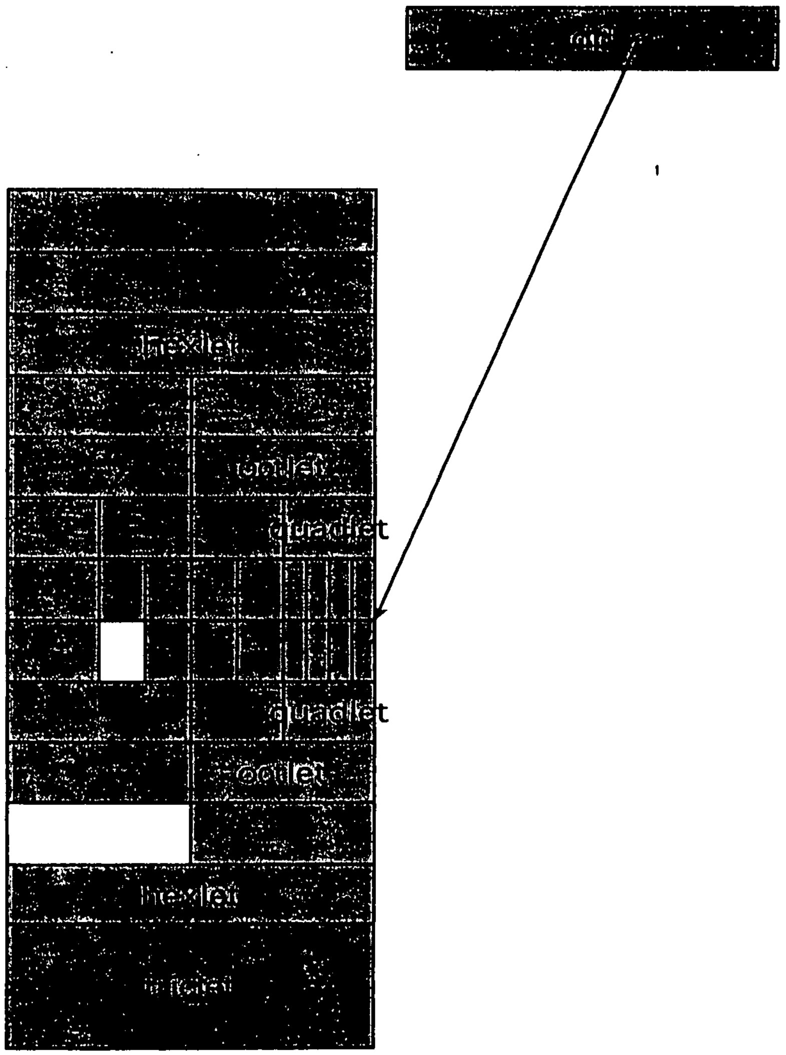
- Gateway
  - level 0 to 2
  - secure entry
  - data pointer
  - stack pointer
- Code
  - LI64LA dp=dp,off
  - LI64LA lp=dp,0
  - B.GATE (lp=dp,lp)

  - LI64LA dp=dp,8
  - SI64LA sp,dp,off
  - LI64LA sp=dp,off



Data pointer

- Memory pool for literals, statics
- procedures may share pool
- items sorted by size
- smallest items near dp
- All items aligned to size



MicroUnity

Procedure call conventions

- Compatible with dynamic linking
- Register 63 (sp) is stack pointer
- Stack space allocated for parameters by caller
- Up to 8 parameters passed in registers 2-9
- Register 0 (lp) loaded with procedure address
- Register 1 (dp) loaded with data pointer
- To enter: BLINK lp=lp
- Register 2 contains return value
- To return: B lp

Procedure Call Structure

- Caller (non-leaf):

```
ADDI      sp,-size       # allocate stack space
SI64LA    lp,sp,off      # save link pointer
SI64LA    dp,sp,off      # save data pointer
  ...                    # use data pointer
B.LINK.I  callee         # call procedure with shared dp
  ...                    # use data pointer
LI64LA    lp=dp,off      # load callee code address
LI64LA    dp=dp,off      # load callee data pointer
B.LINK    lp             # call procedure
  ...                    # data pointer not available
LI64LA    dp=sp,off      # reload data pointer
  ...                    # use data pointer
LI64LA    lp,sp,off      # reload link pointer
ADDI      sp,size        # deallocate stack space
B         lp             # return to caller
```

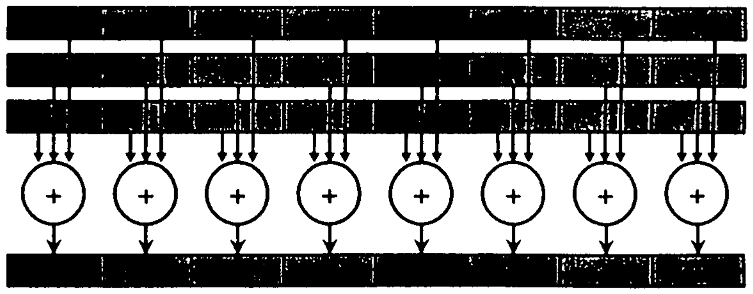
- Callee (leaf):

```
  ...                    # args in reg, use data pointer
B         0              # return to caller
```

MicroUnity

Group Instructions

- Fixed-point operations over 128-bit operands with 8..128 bit symbols
- Add, Subtract, Set-conditional
- 3-operand Add/Subtract
- Add/Subtract Halve, Limiting
- Boolean: 3-operand, MUX
- Shift left immediate add
- Compare



Group triple operand

- Reduces latency for common arithmetic operations
- Group triple add/subtract
  - $rd_{128} = rd_{128} \pm rc_{128} + rb_{128}$
  - 8-128 bit symbols
- Group shift 1-4 and add/subtract
  - matches load/store with shifted index
- Group triple boolean immediate
  - $rd_i = f(rd_i, rc_i, rb_i)$, $i=0..127$
  - 8 immediate bits specify f

MicroUnity

Typical boolean functions

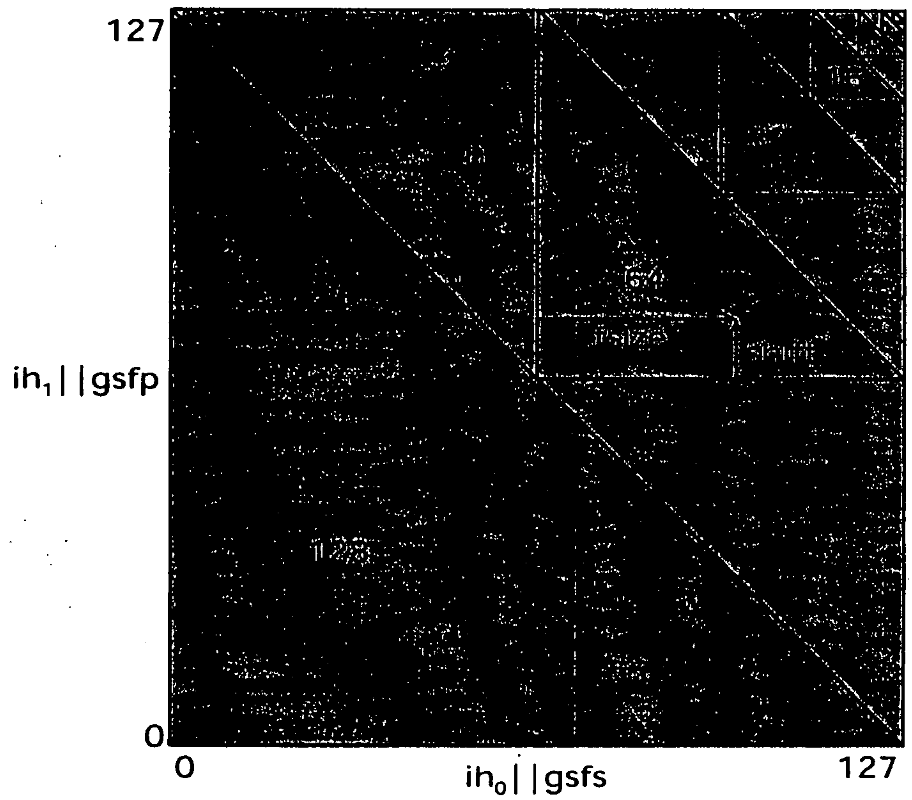
■ dcb	10000000	128
■ dc|b	11101010	234
■ d|c|b	11111110	254
■ d?c:b	11001010	202
■ d^c^b	10010110	150

X: Crossbar Instructions

- Deposit, Withdraw
- Extract, Expand, Compress
- Swizzle, Select, Shuffle
- Shift
- Shift-Merge
- Rotate
- Wide Switch

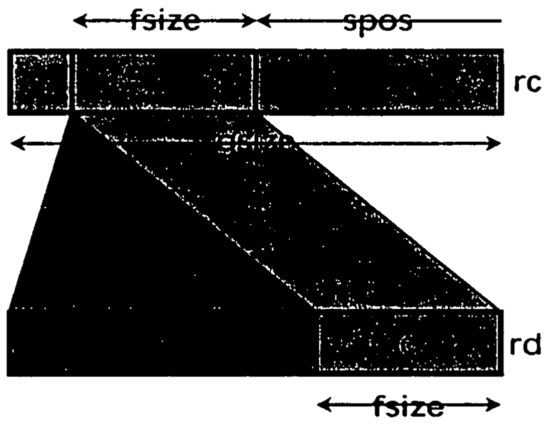
Crossbar field

- fsize, shift (or spos/dpos)

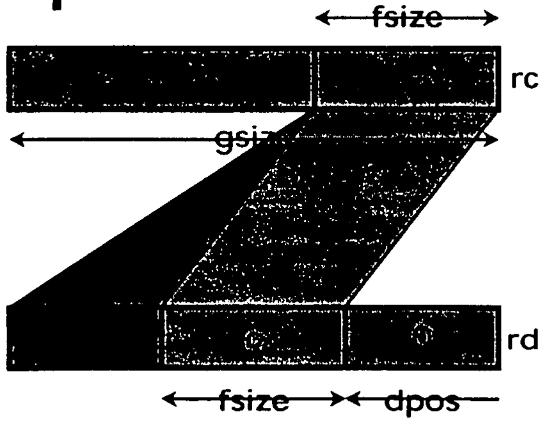


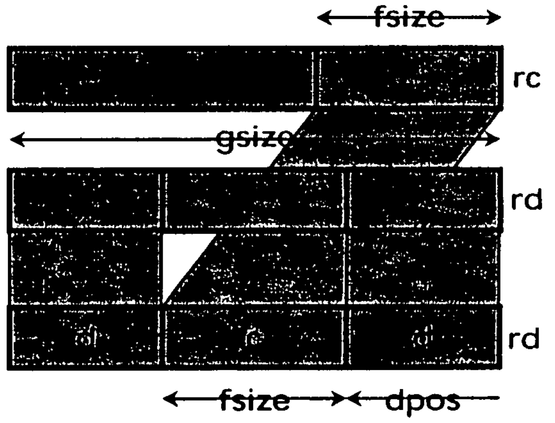
Crossbar field

- withdraw



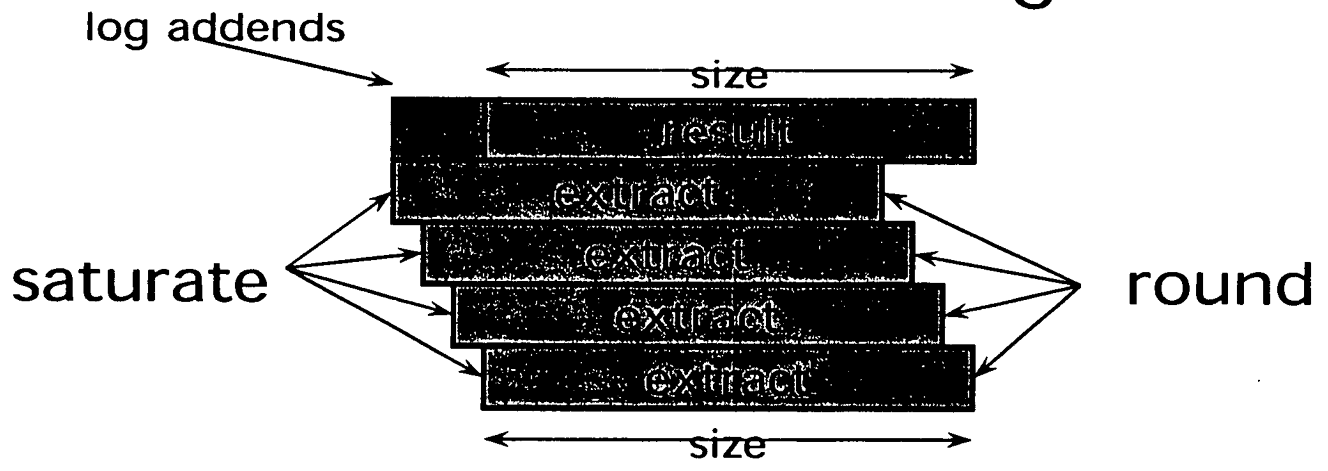
- deposit





Crossbar extract control

- immediate control fields
  - 2 size 8, 16, 32, or 64 bits
  - 1 saturate signed, unsigned
  - 2 round floor, ceil, zero, even
  - 2 shift 0-3 bits from right



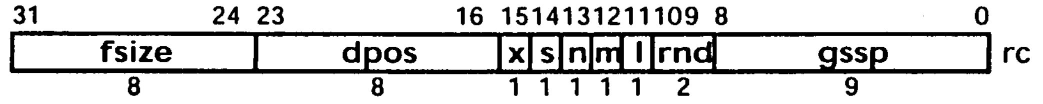
Crossbar extract

- $rd_i = (ra_{128} \,||\, rb_{128})_{f(rc_{32}, i)}$, $i=0..127$
- extract w/register operand control
- register specifies:

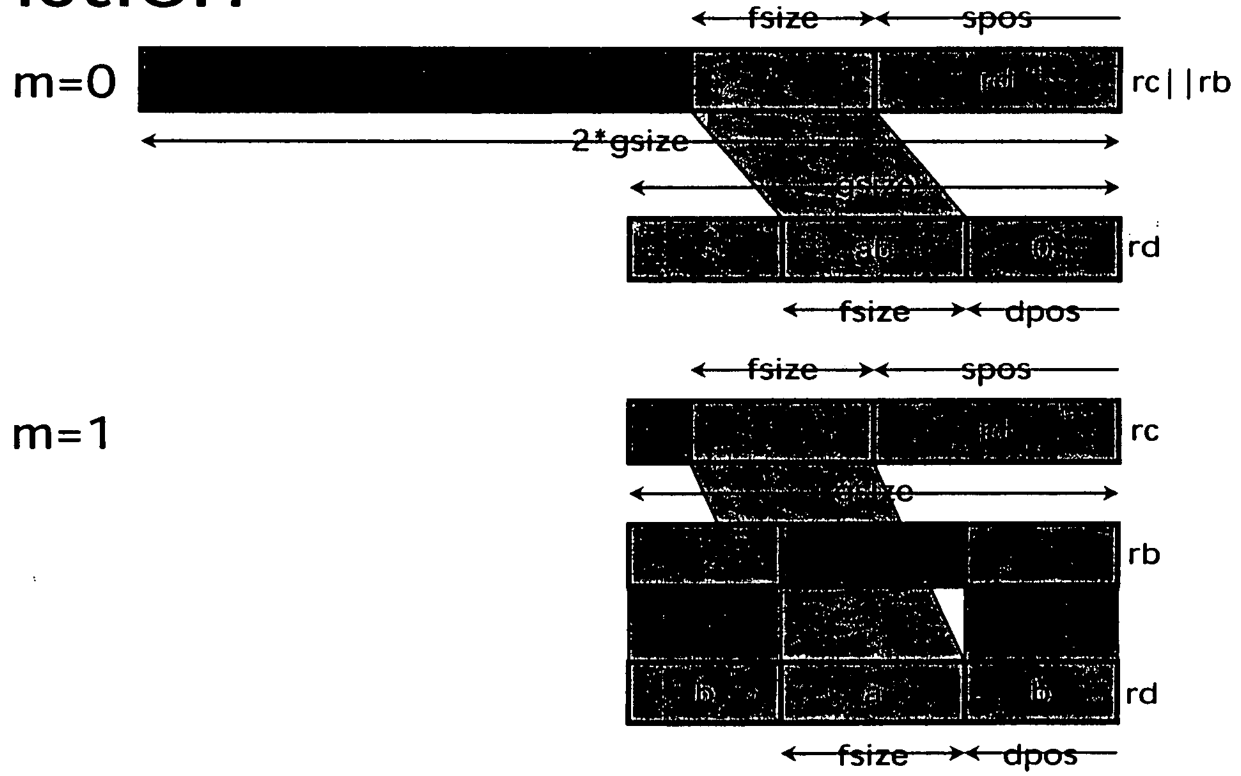
8	fsize	field size
8	dpos	destination position
9	gssp	group size and source position
1	s	signed vs unsigned
1	n	(real vs complex)
1	m	extract vs merge (or mixed sign)
1	l	saturation vs truncation
2	rnd	rounding

Crossbar extract control

- layout

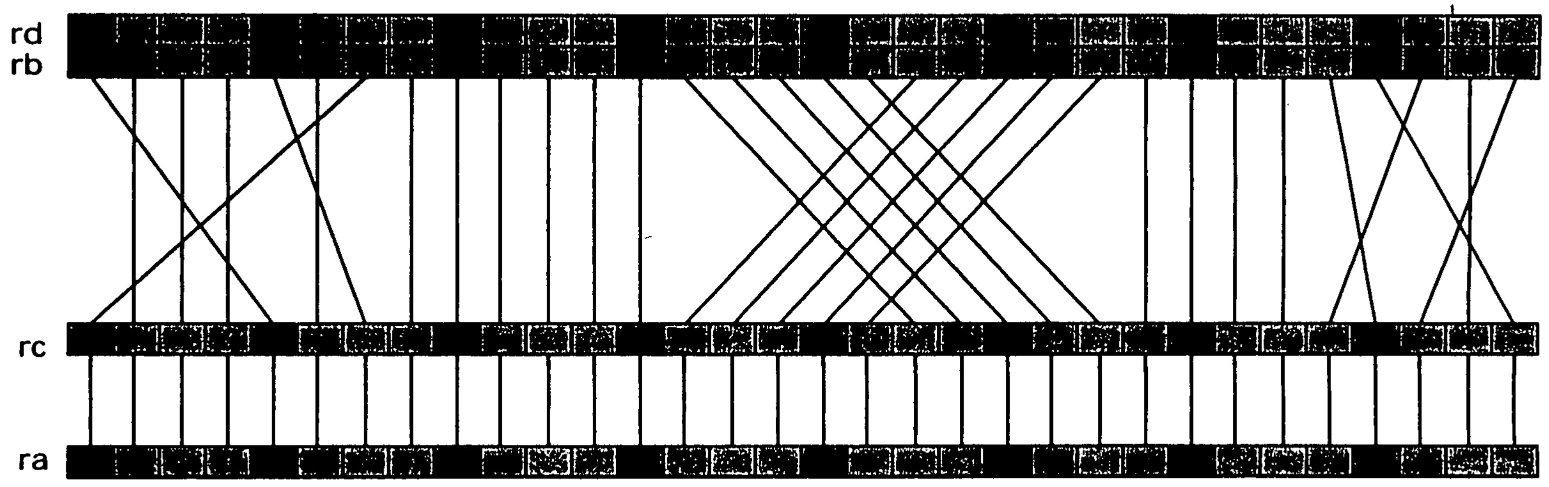


- function



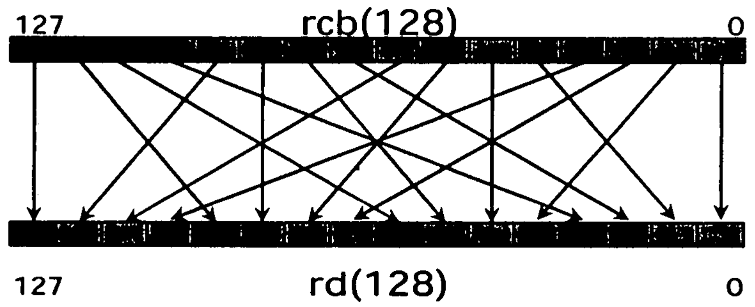
Crossbar Select bytes

- X.SELECT.8 ra=rc,rd,rb



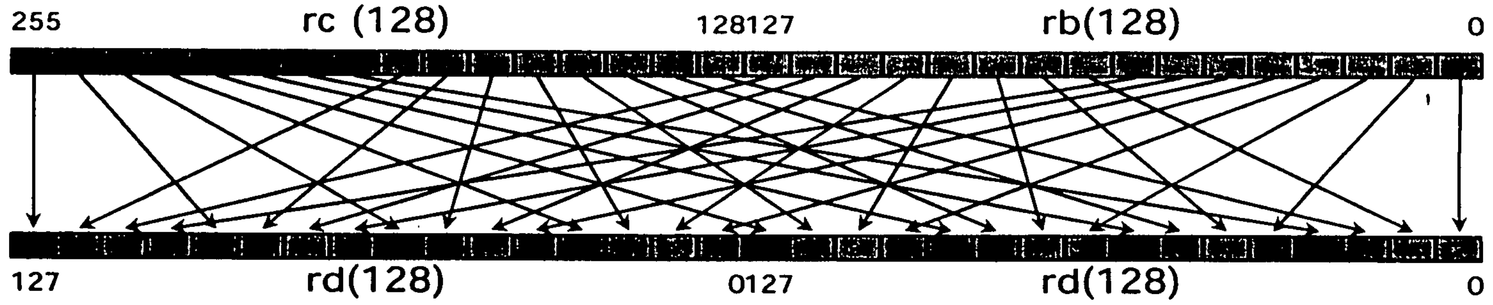
4-way shuffle bytes within hexlet

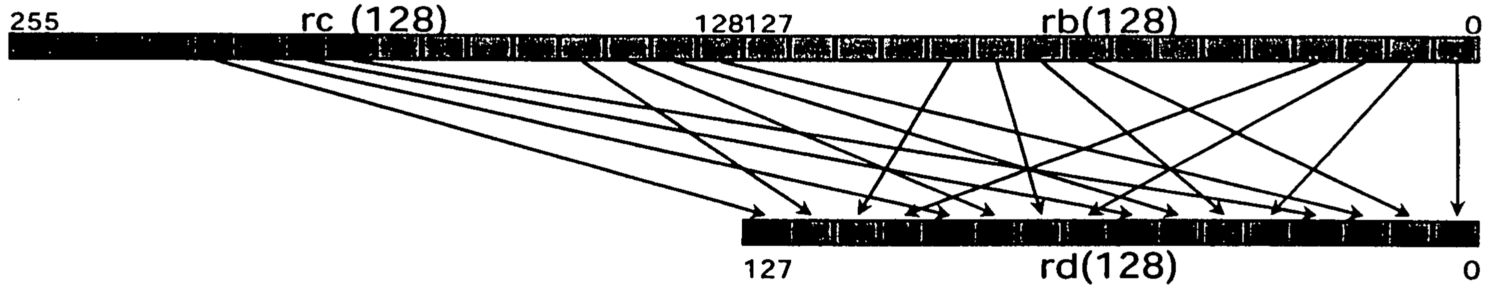
- XSHUFFLEI.128 rd=rcb,8,4



4-way shuffle bytes within triclet

- XSHUFFLEI.128 rd=rc,rb,8,4





MicroUnity

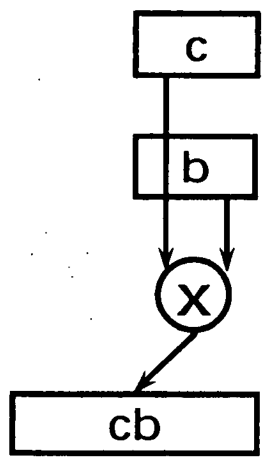
Ensemble Instructions

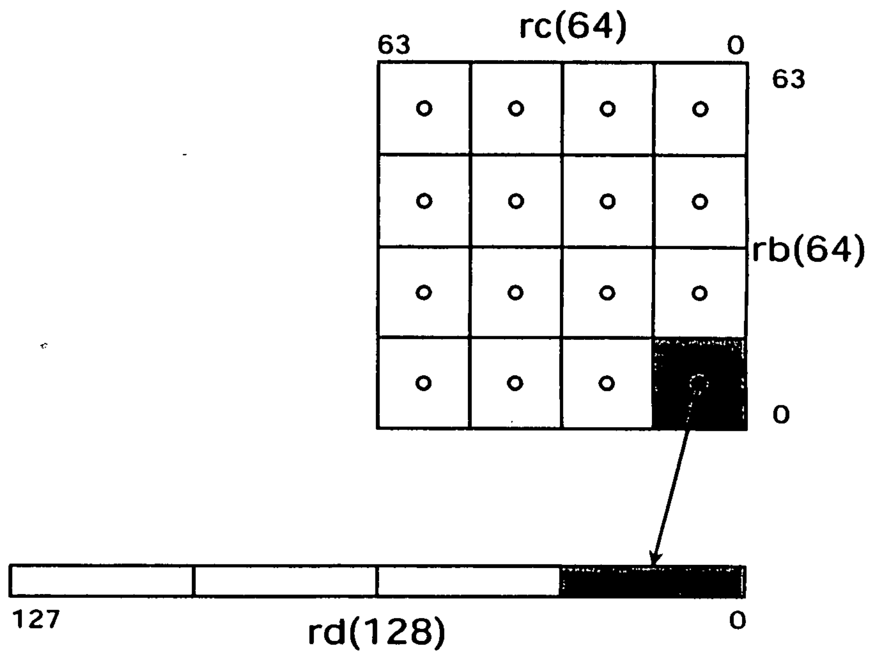
- Multiply
  - Fixed-point
    - size-doubling
    - extract
  - Floating-point
  - Complex
  - Polynomial
  - Galois Field

  - Convolve
  - Multiply-add
  - Scale-add
  - Multiply-sum
- Floating-point
  - Add, Subtract, Divide, Sum
  - Inflate, Deflate, Float, Sink
  - Reciprocal Estimate
  - Reciprocal Square Root Estimate
- Fixed-point
  - Sum
  - Log-most

Multiply

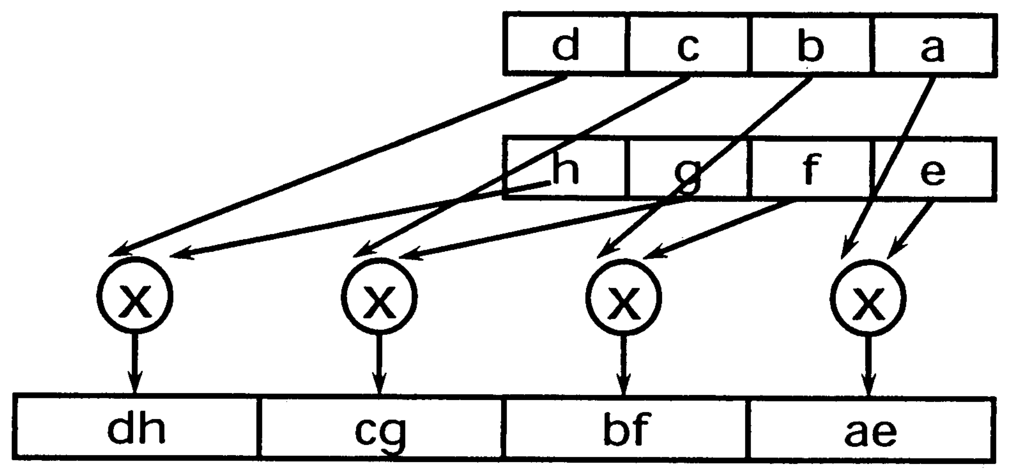
- $rd_{32} = rc_{16} * rb_{16}$





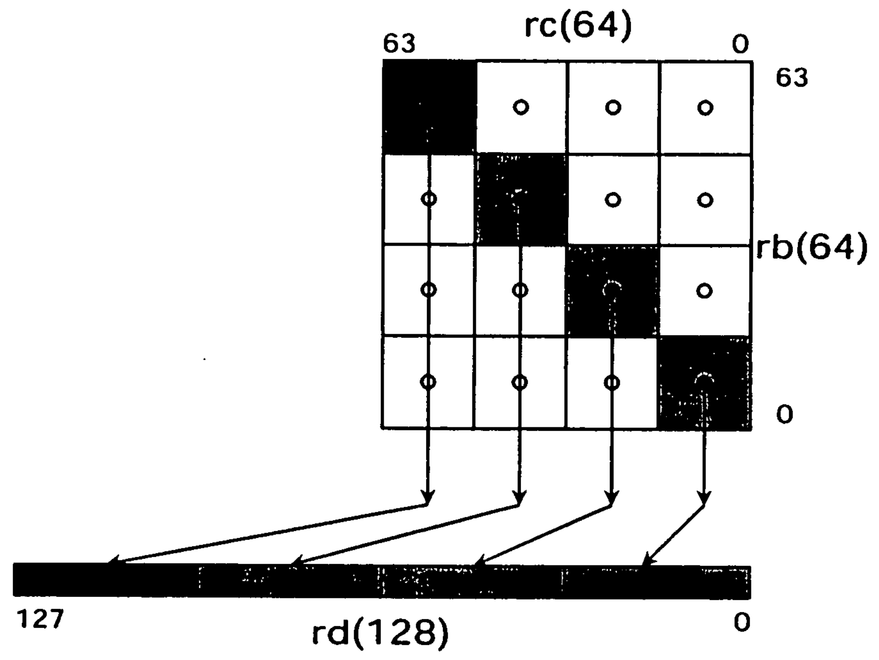
Ensemble multiply

- $rd_{128} = rc_{64} * rb_{64}$



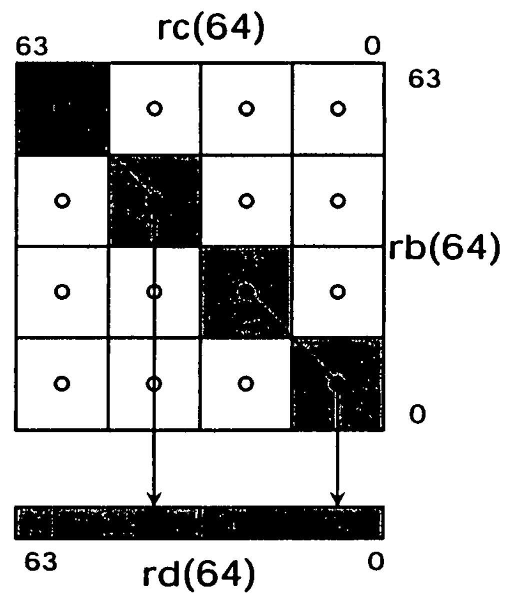
Ensemble multiply

- $rd_{128} = rc_{64} * rb_{64}$



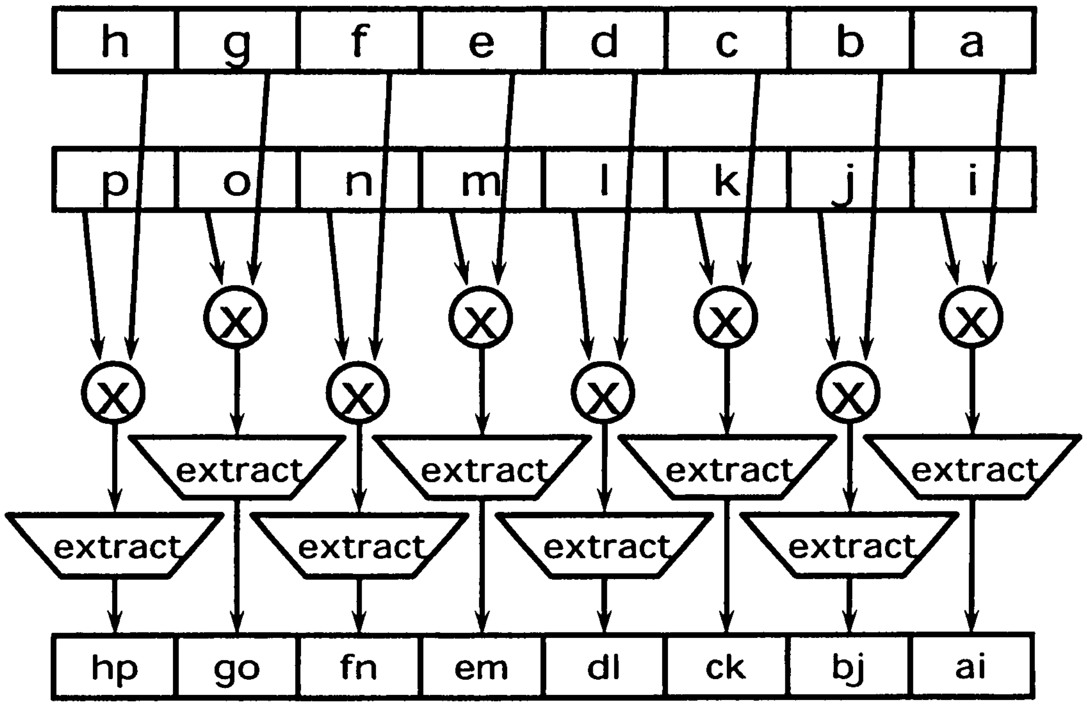
MMX PMADDWD

- $rd_{128} = rc_{64} * rb_{64}$



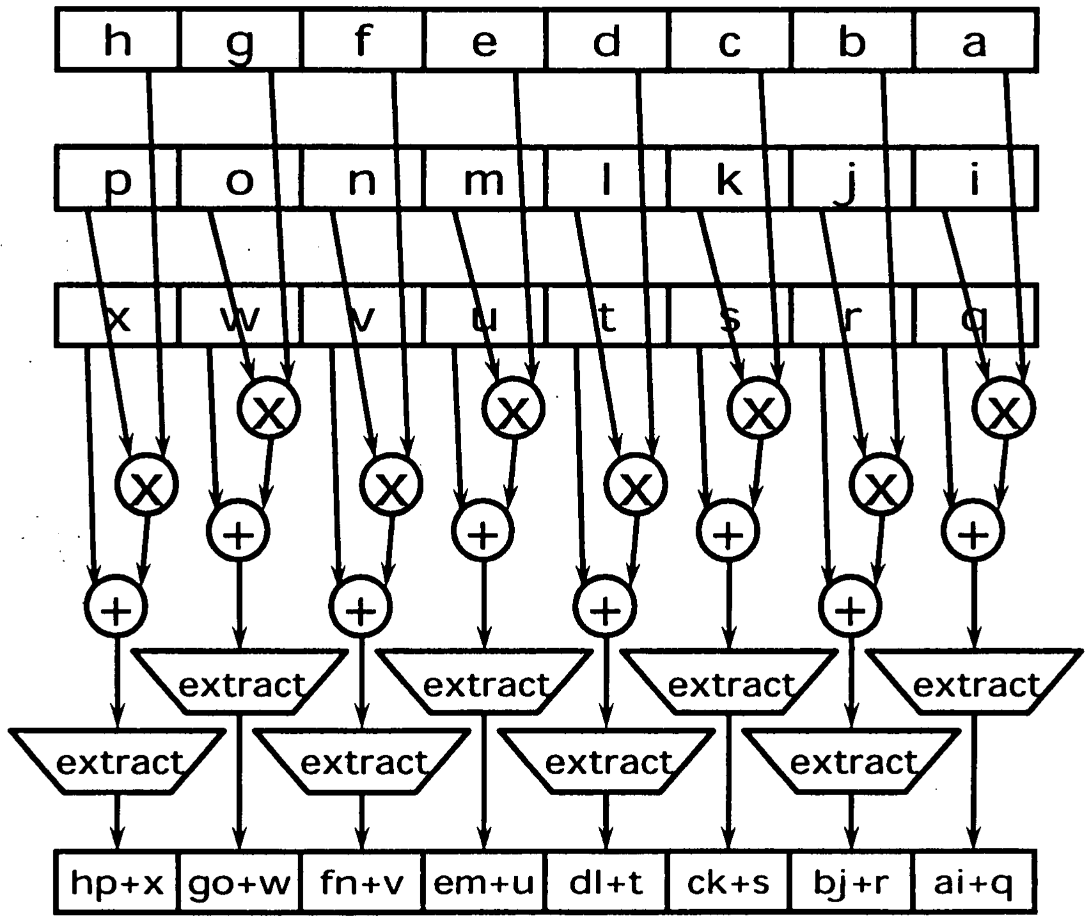
Ensemble multiply extract

- $rd_{128} = rc_{128} * rb_{128}$



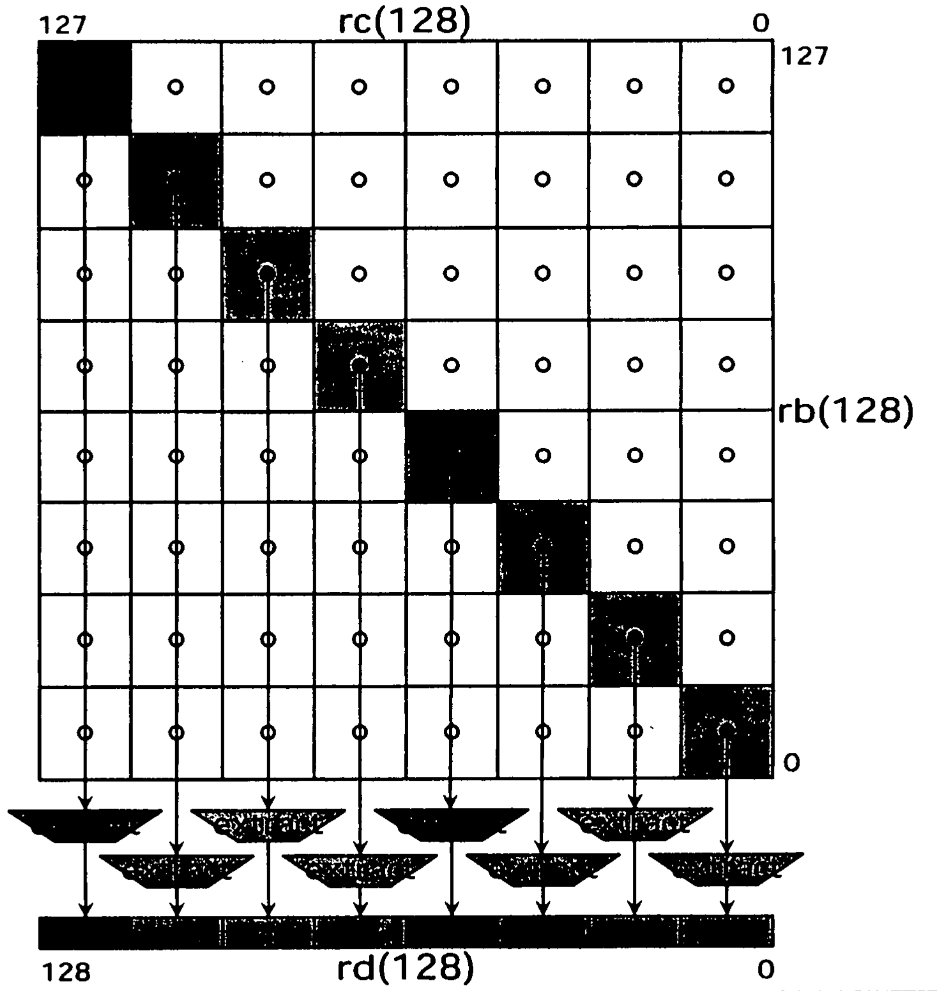
Ensemble multiply add extract

- $rd_{128} = rc_{128} * rb_{128} + rd_{128}$



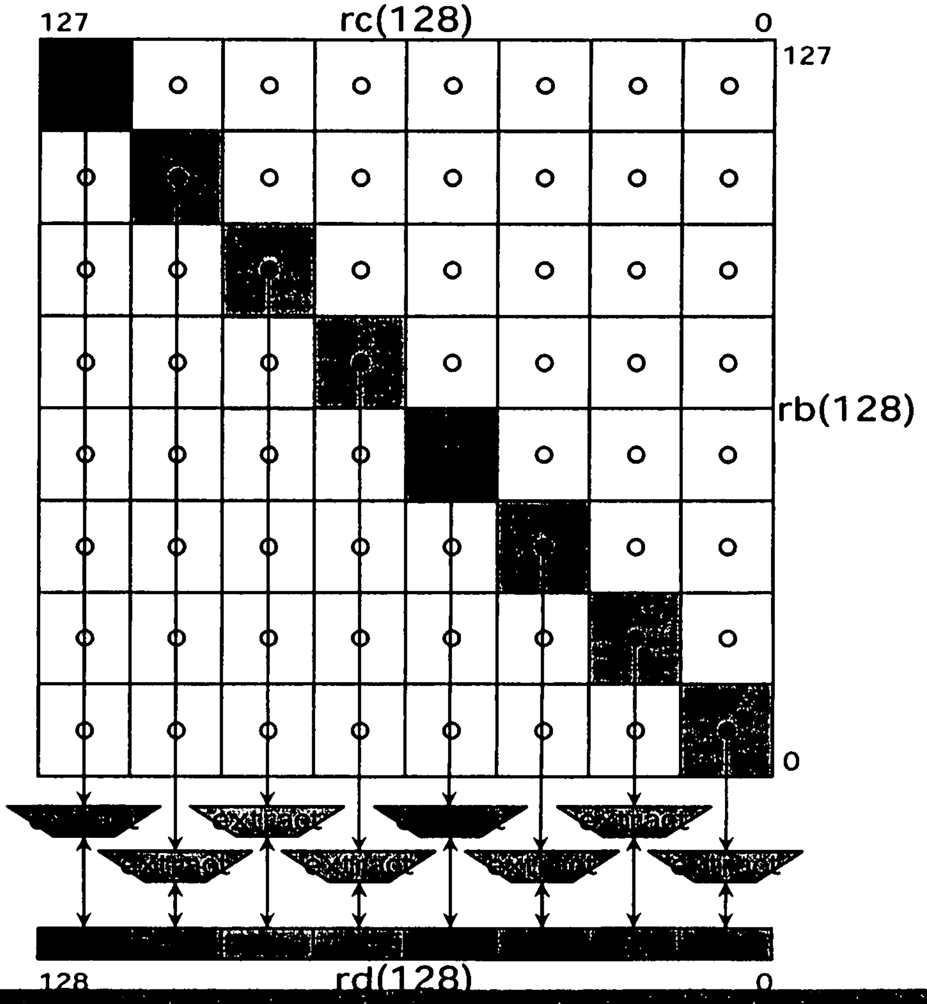
Ensemble multiply extract

- $rd_{128} = rc_{128} * rb_{128}$



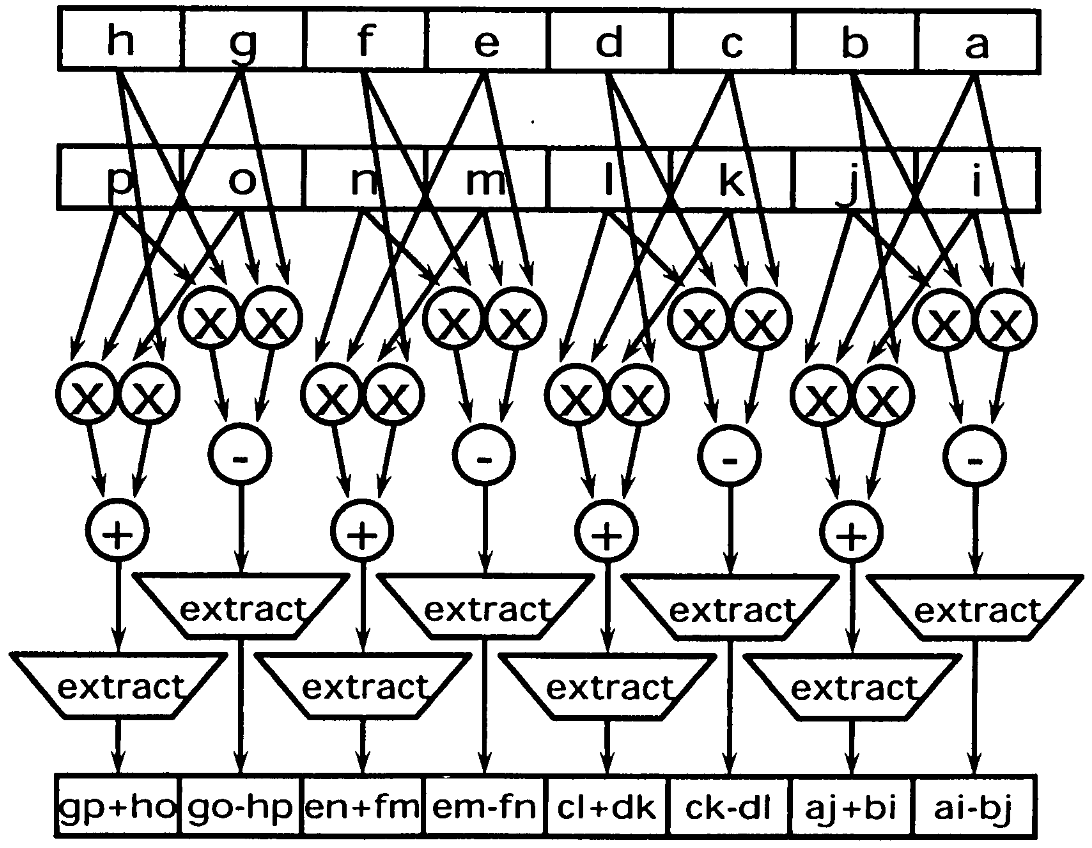
Ensemble multiply add extract

- $rd_{128} = rc_{128} * rb_{128} + rd_{128}$



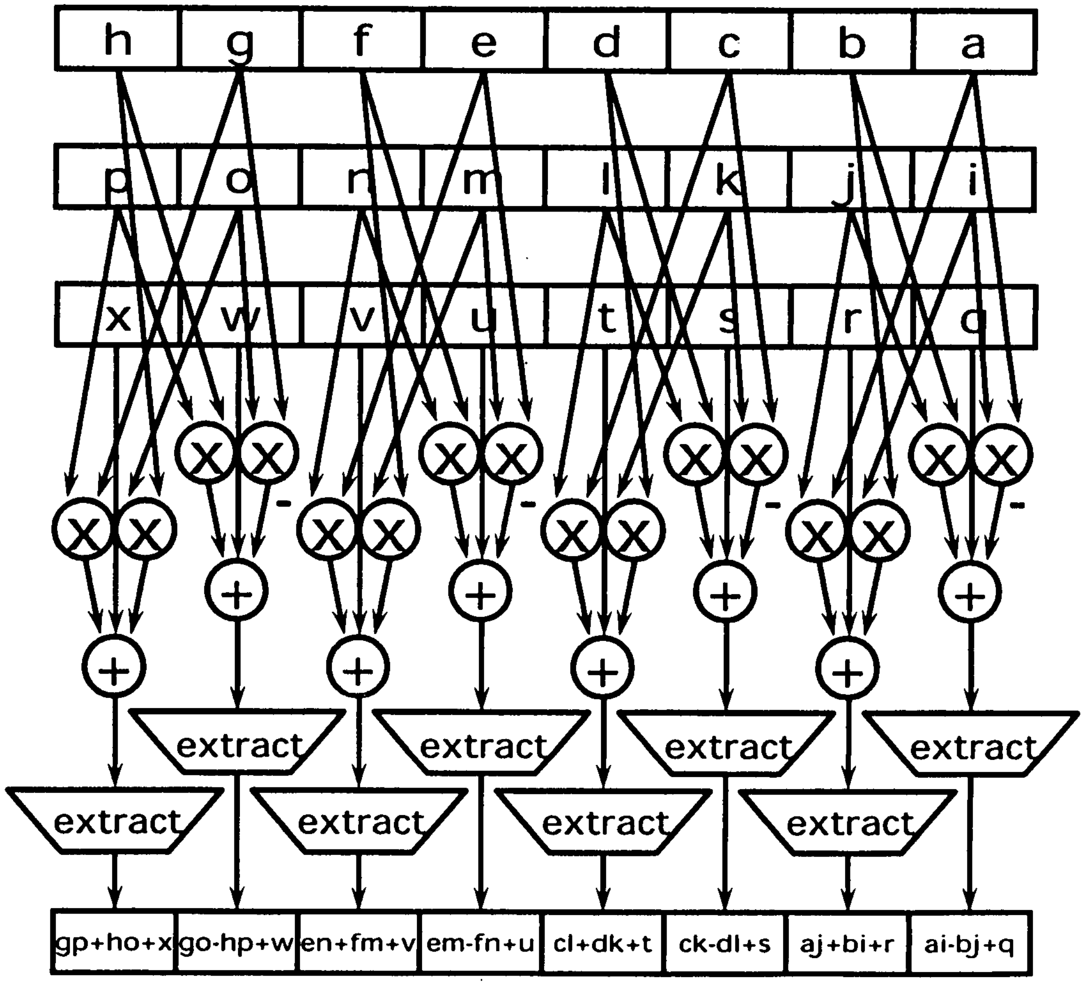
Ensemble multiply extract complex

- $rd_{128} = rc_{128} * rb_{128}$



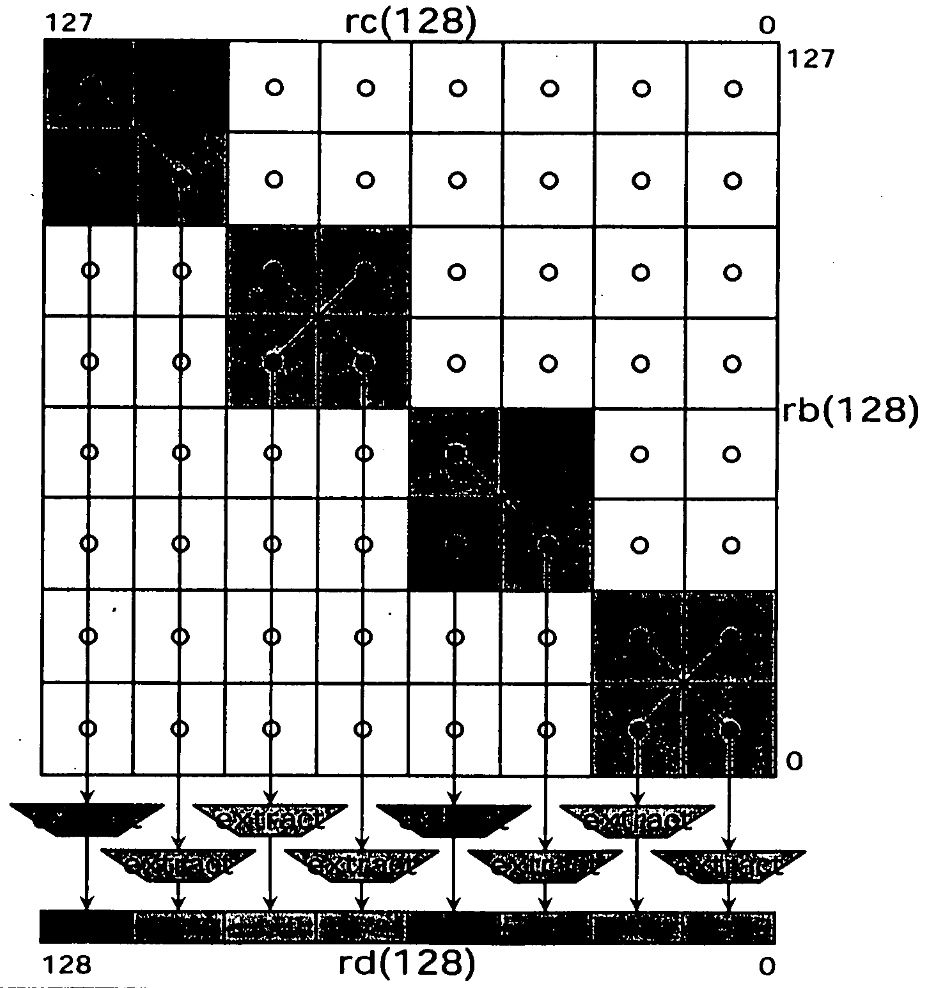
Ensemble multiply add extract complex

- $rd_{128} = rc_{128} * rb_{128} + rd_{128}$



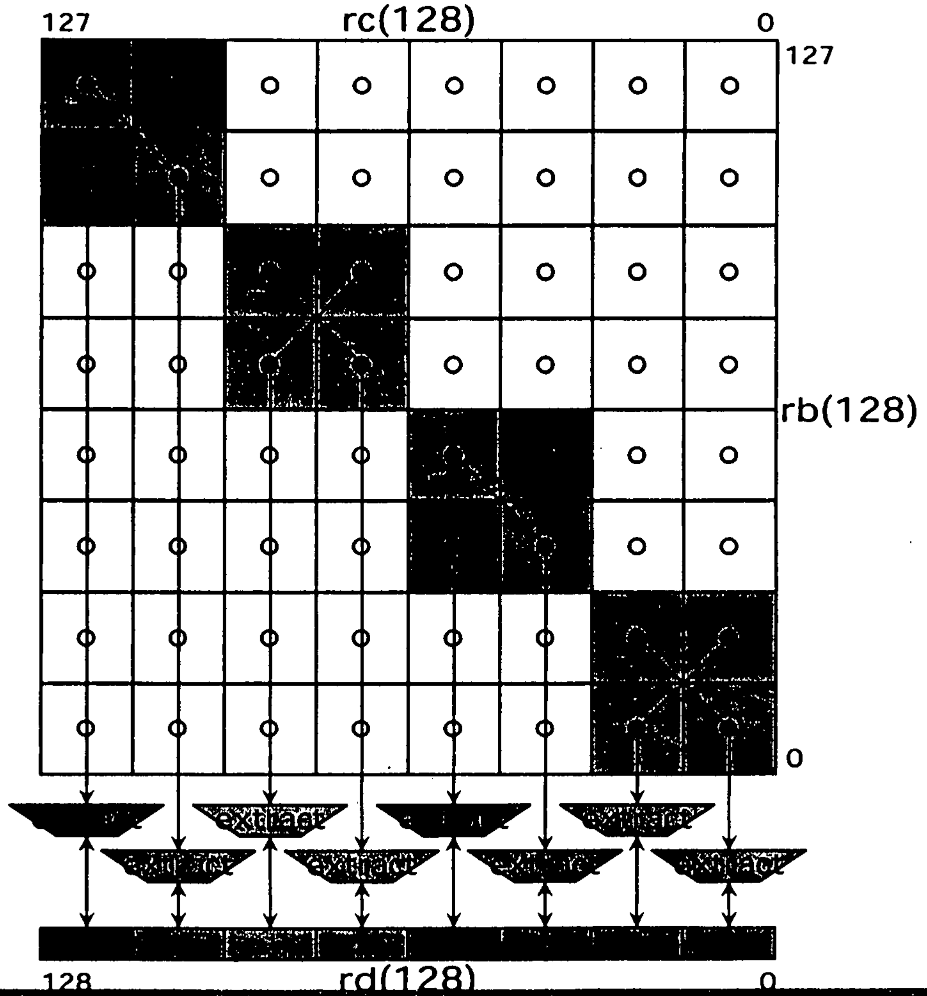
Ensemble multiply extract complex

- $rd_{128} = rc_{128} * rb_{128}$



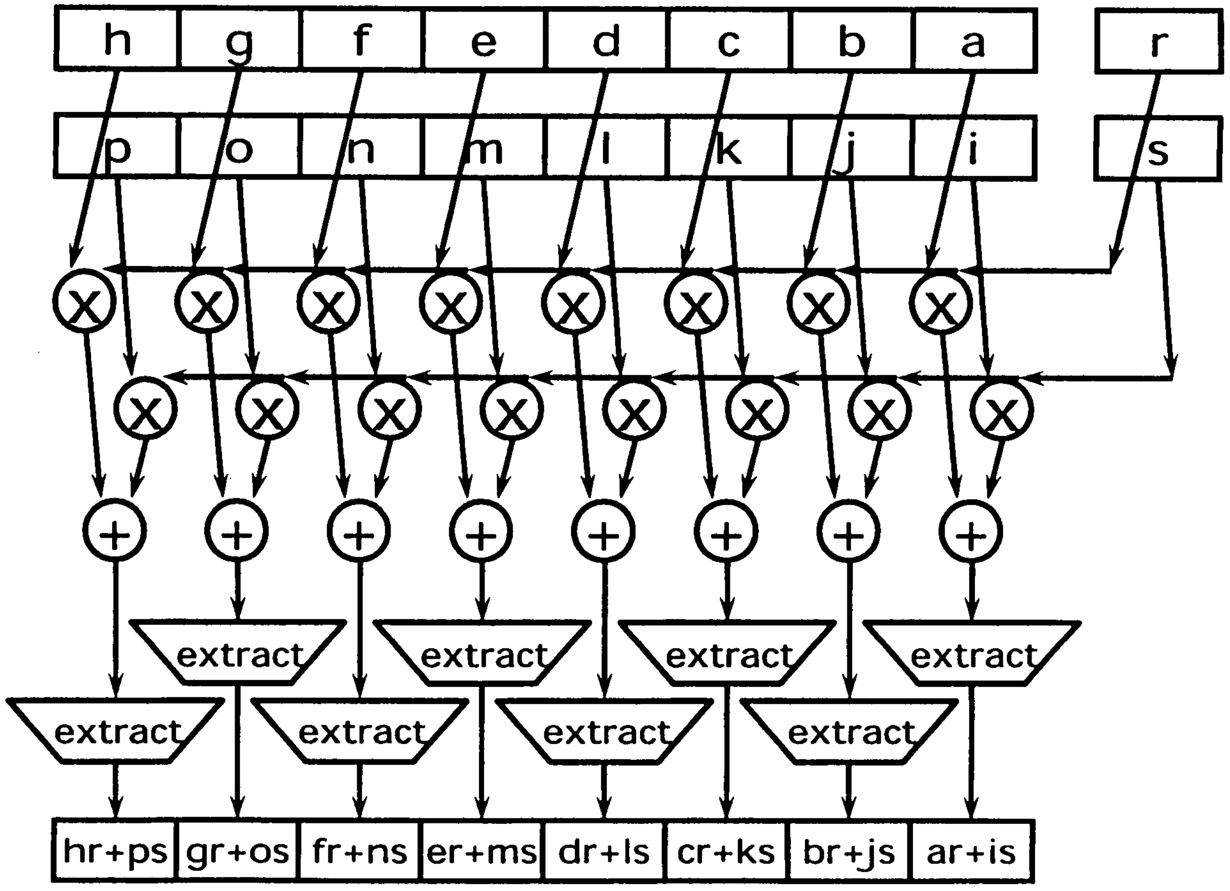
Ensemble multiply add extract complex

■ $rd_{128} = rc_{128} * rb_{128} + rd_{128}$



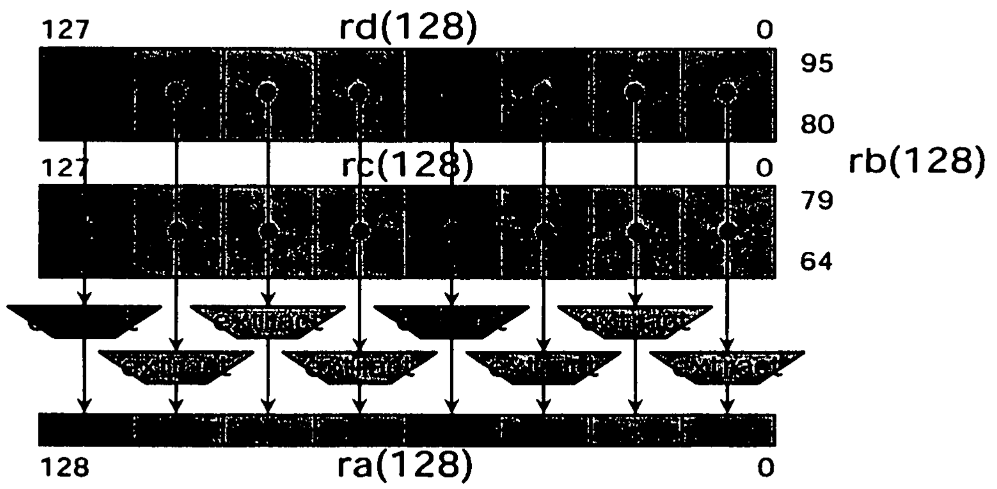
Ensemble scale add extract

- $ra_{128} = rd_{128} * rb_{size} + rc_{128} * rb_{size}$



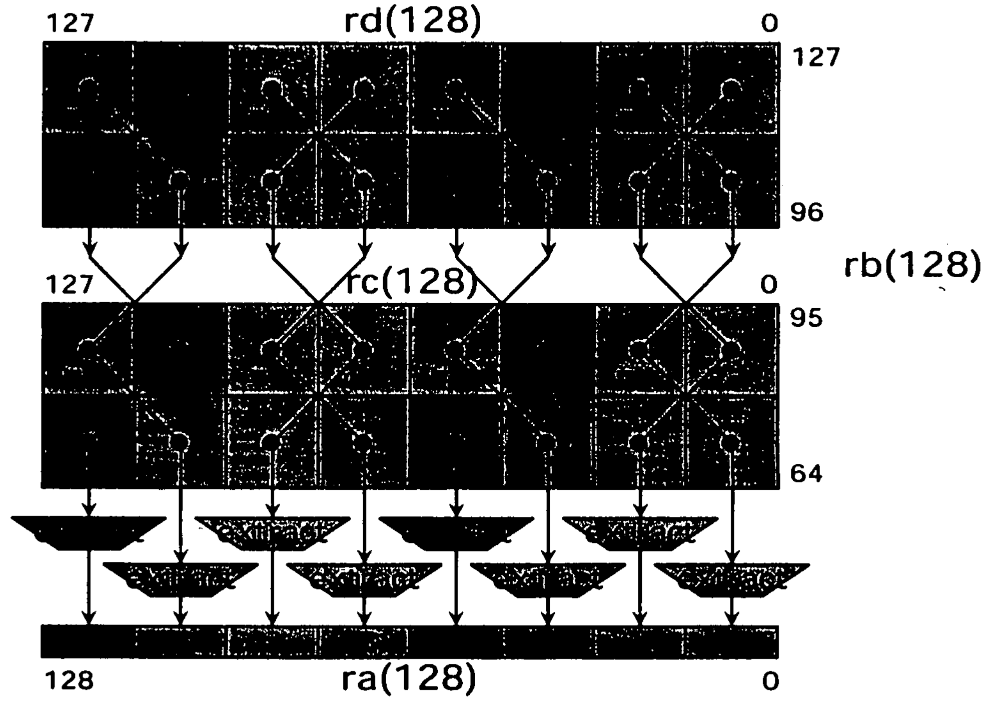
Ensemble scale add extract

- $ra_{128} = rd_{128} {}^{*} rb_{size} + rc_{128} {}^{*} rb_{size}$



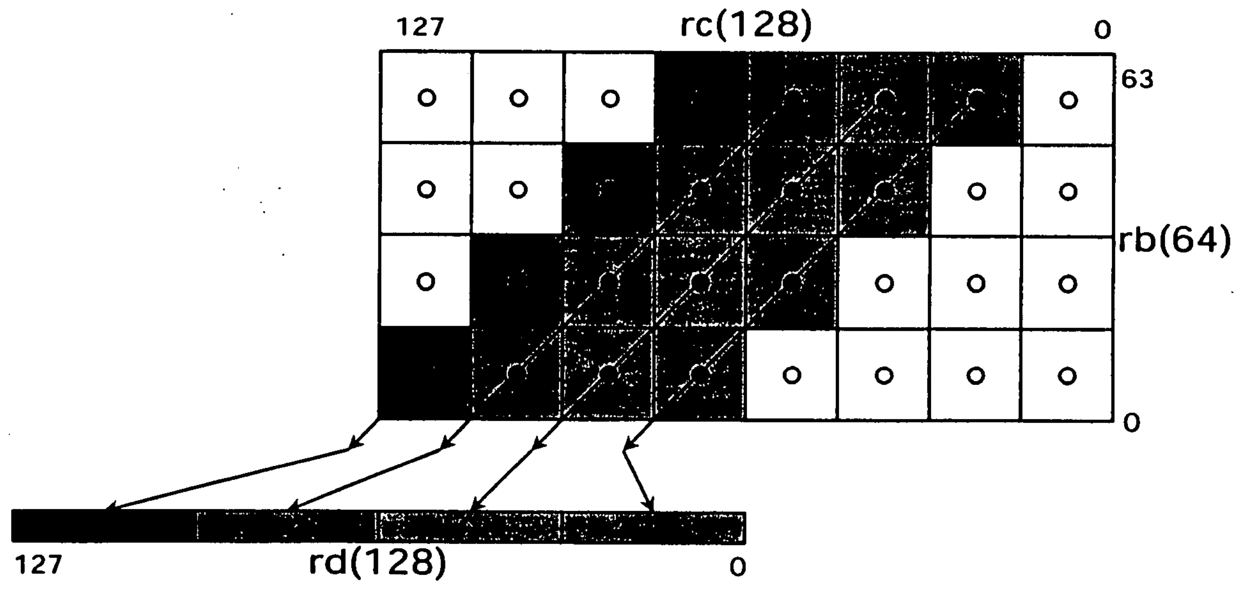
Ensemble scale add extract complex

- $ra_{128} = rd_{128} * rb_{size*2} + rc_{128} * rb_{size*2}$



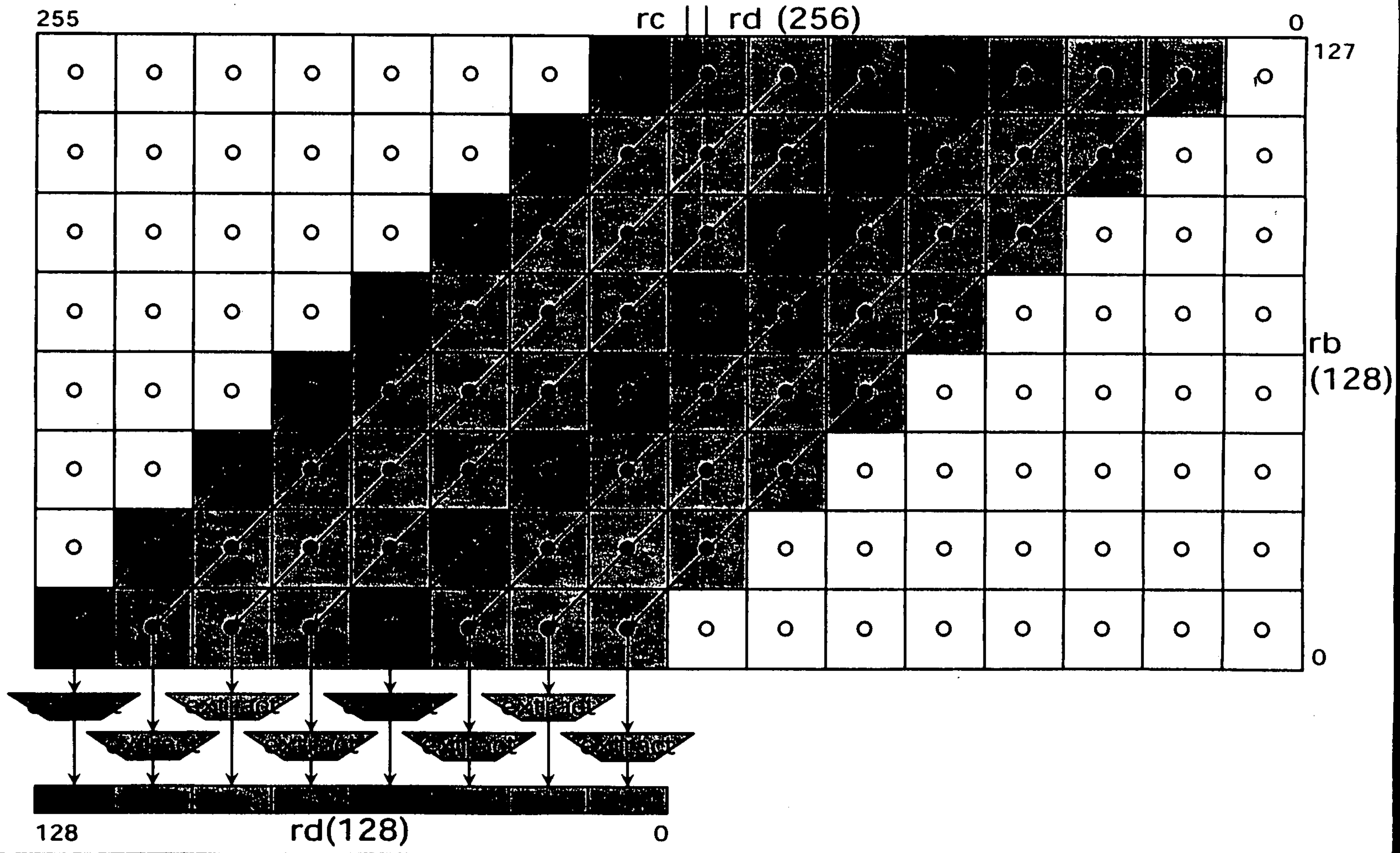
Ensemble convolve

- $rd_{128} = rc_{128} * rb_{64}$



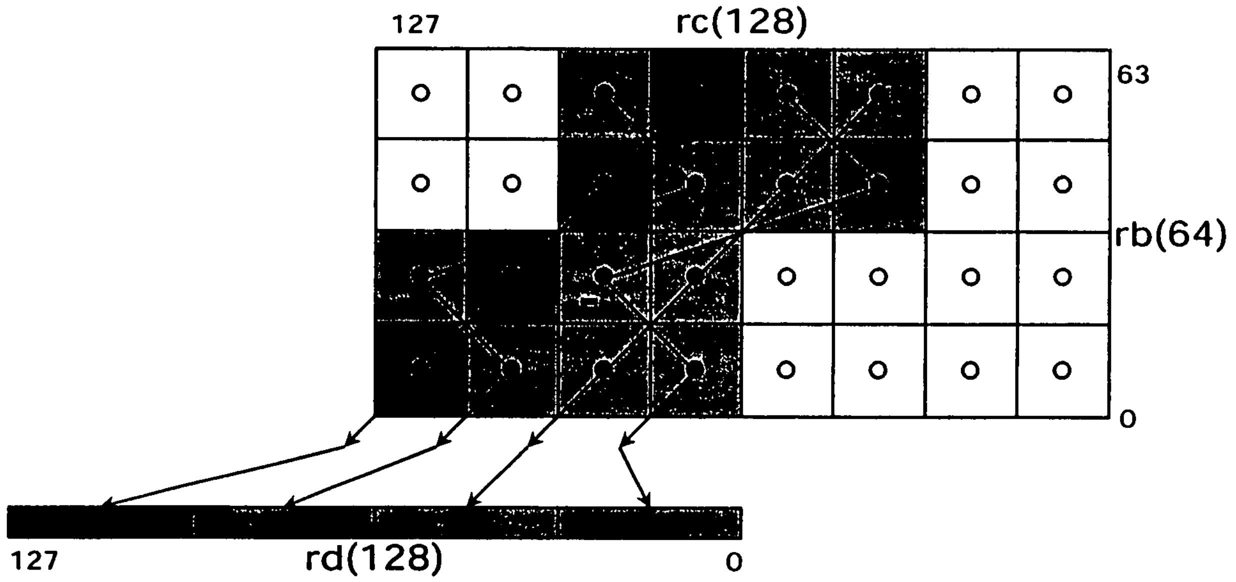
Ensemble convolve extract

- $rd_{128} = (rd \| rc)_{256} * rb_{128}$



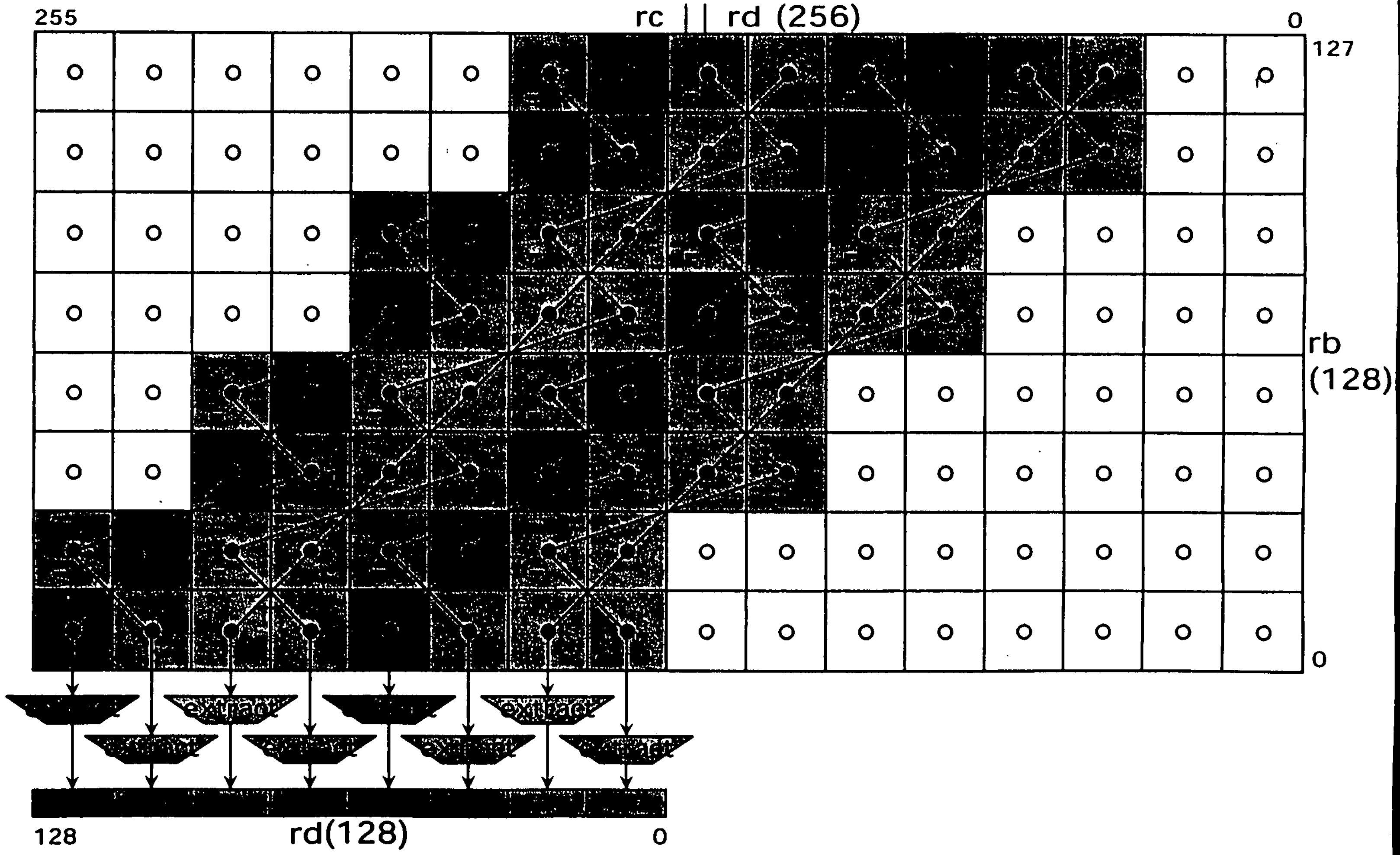
Ensemble convolve complex

- $rd_{128} = rc_{128} * rb_{64}$



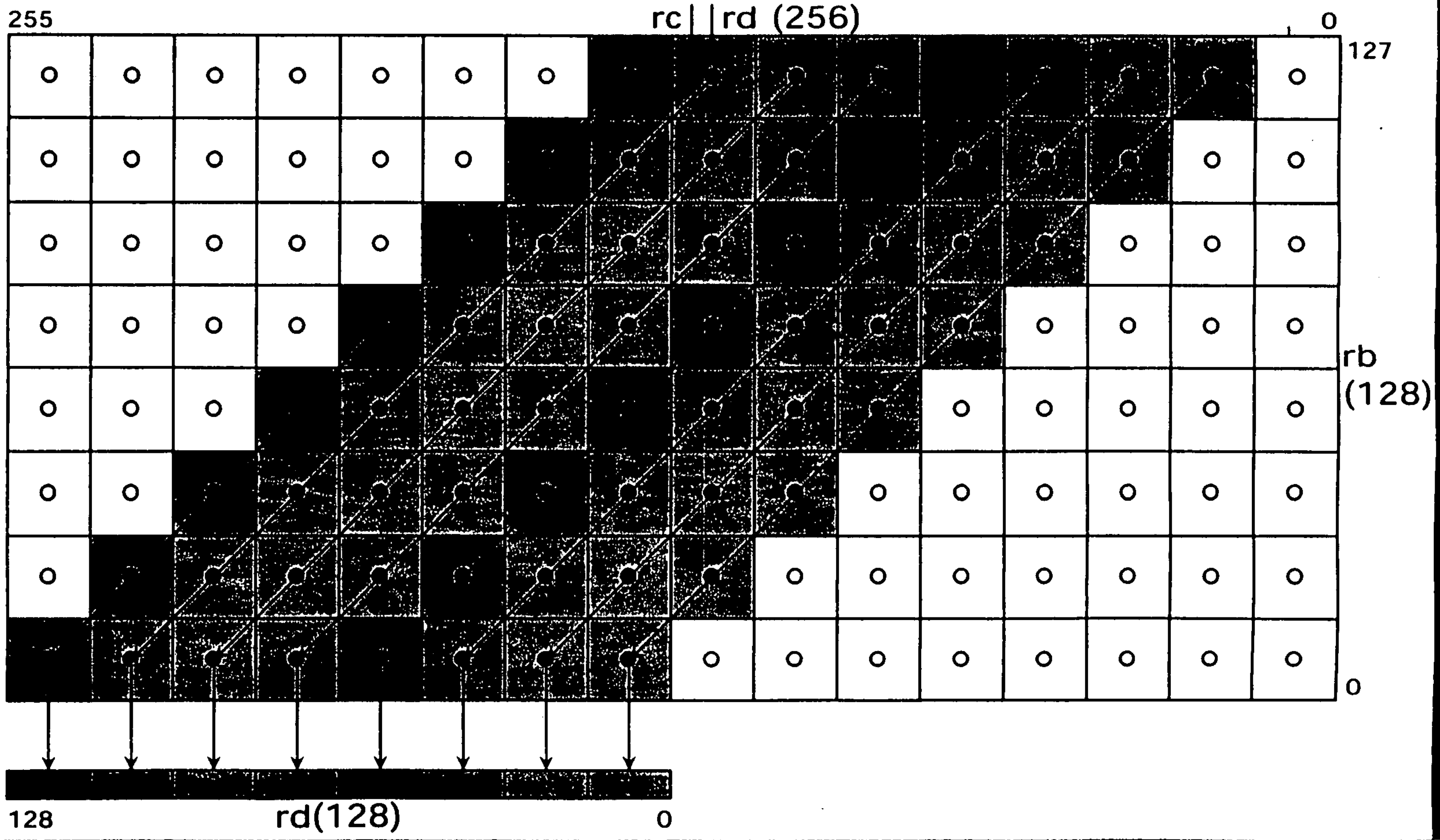
Ensemble convolve extract complex

- $rd_{128} = (rc||rd)_{256} * rb_{128}$



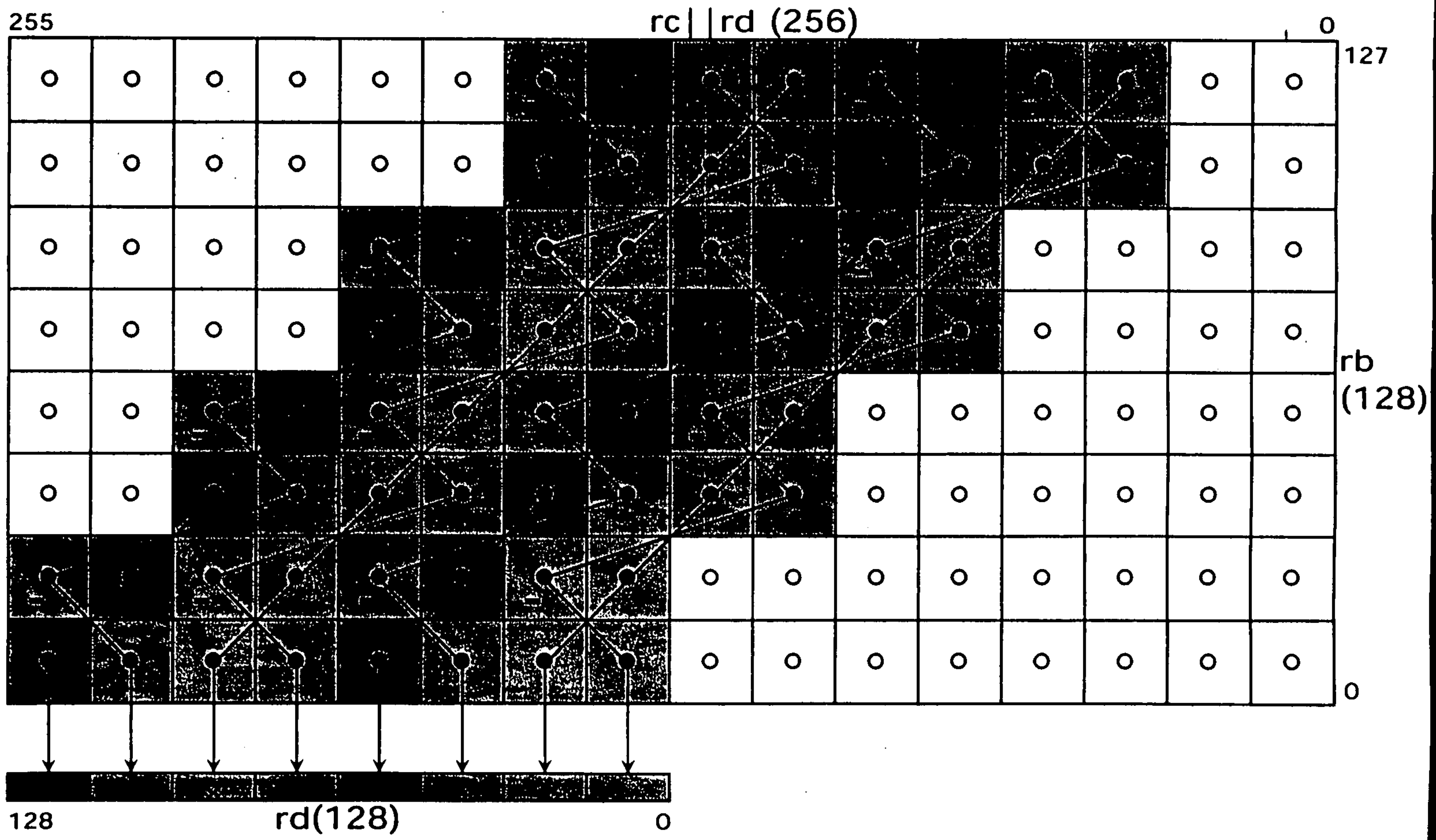
Ensemble convolve floating-point

- $rd_{128} = (rc||rd)_{256} * rb_{128}$



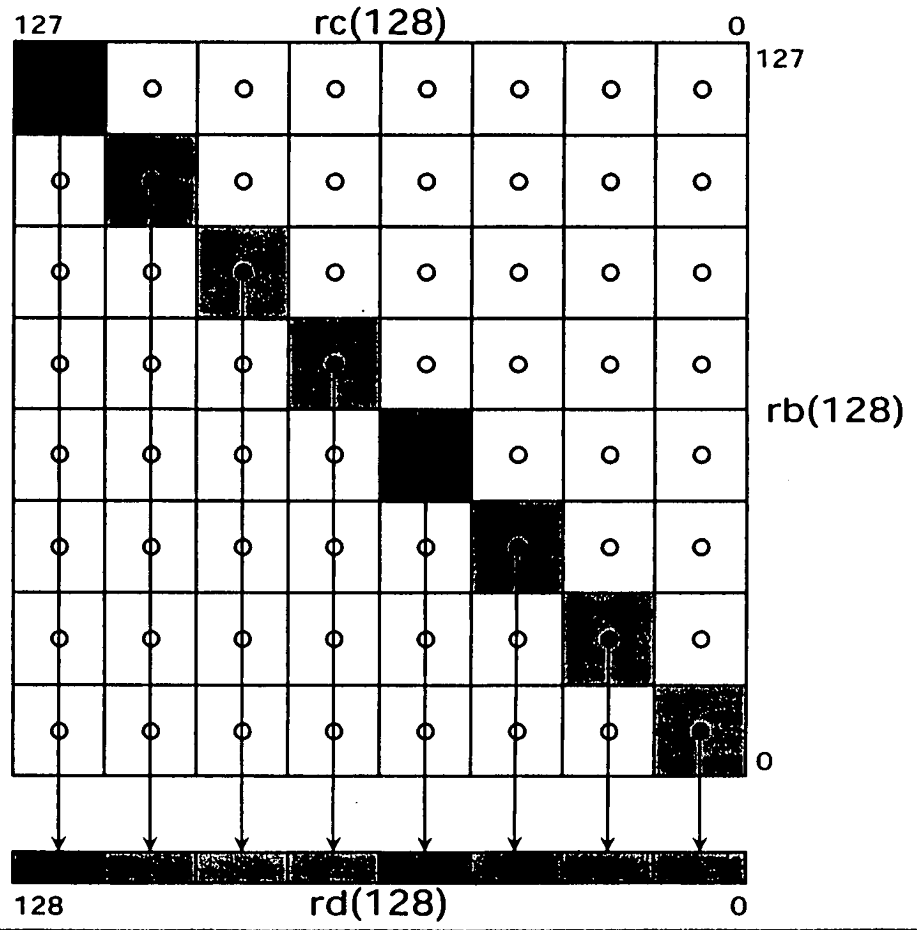
Ensemble convolve complex floating-point

- $rd_{128} = (rc||rd)_{256} * rb_{128}$



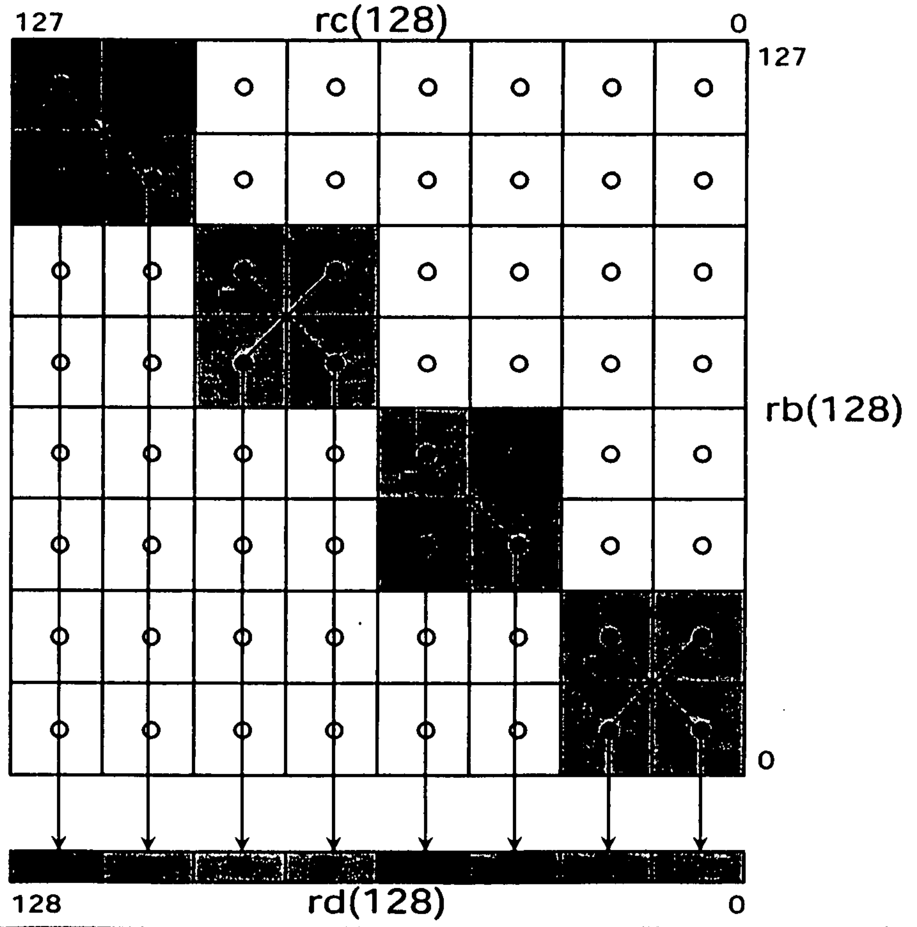
Ensemble multiply floating-point

- $rd_{128} = rc_{128} * rb_{128}$



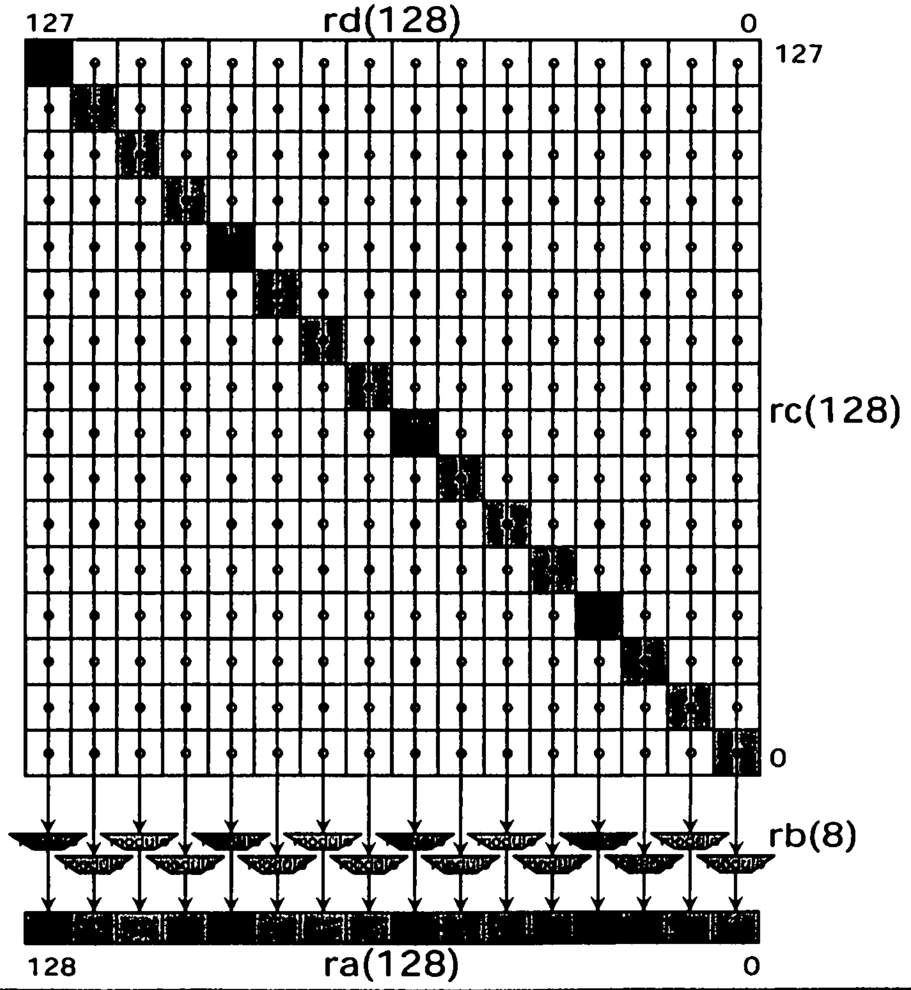
Ensemble multiply floating-point complex

- $rd_{128} = rc_{128} * rb_{128}$



Ensemble multiply Galois

- $ra_{128} = rd_{128} * rc_{128} \bmod rb_8$



MicroUnity

Wide Instructions

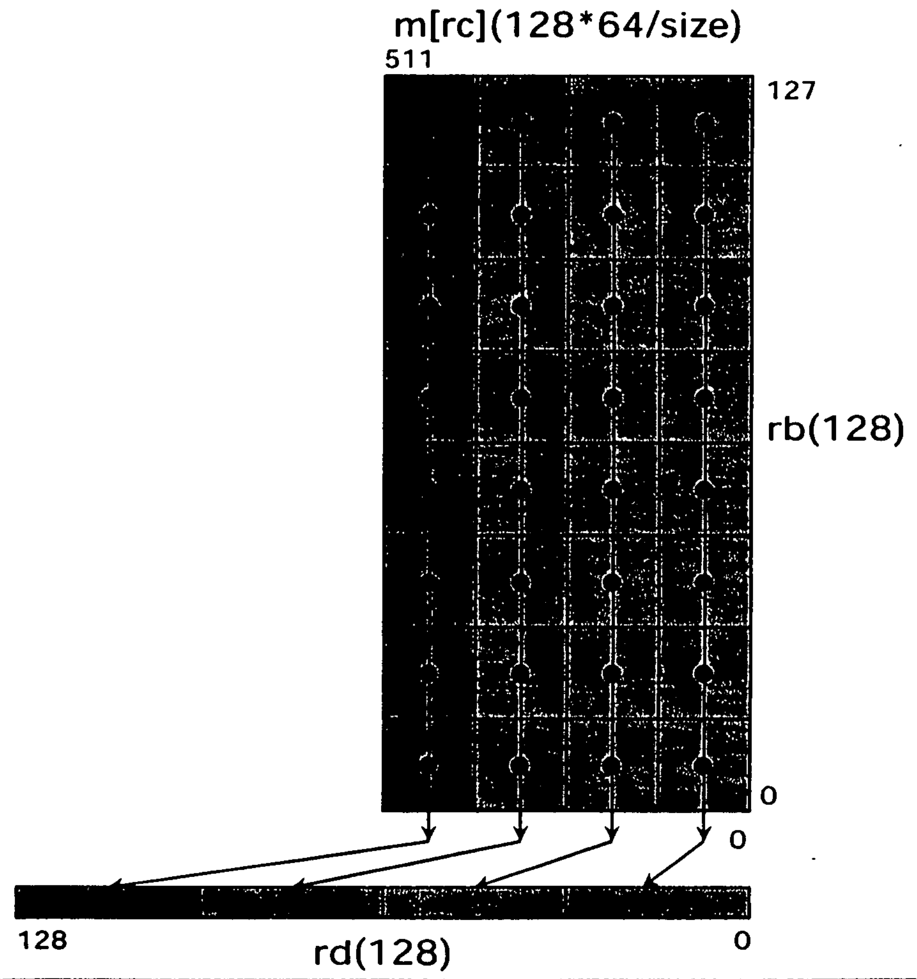
- Wide Multiply Matrix E
- Wide Switch X
- Wide Table L

Wide size and shape

- operations up to 128x128
- full size not always required
- optional bits set in address
  - sets operand size
  - sets operand width
- operand aligned to specified size
- smaller size *may* use fewer cycles
  - to load operand cache
  - to perform operation

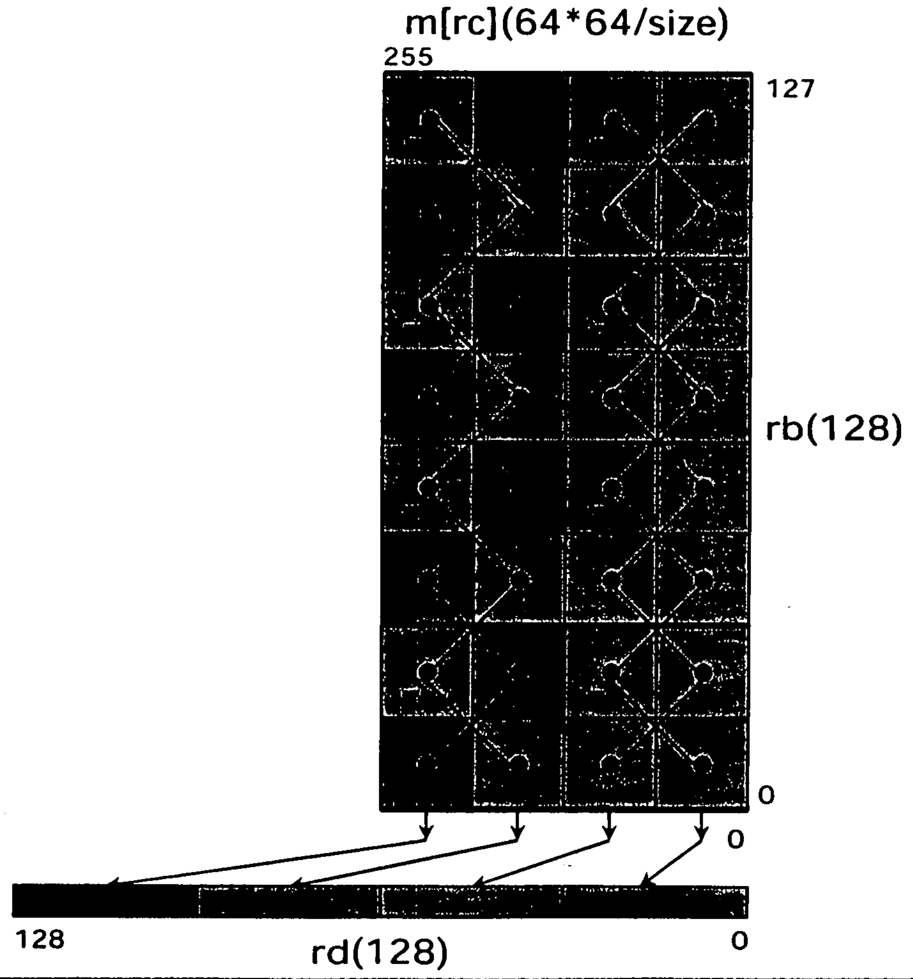
Wide multiply matrix

- $rd_{128} = m[rc]_{(128*64/size)} * rb_{128}$



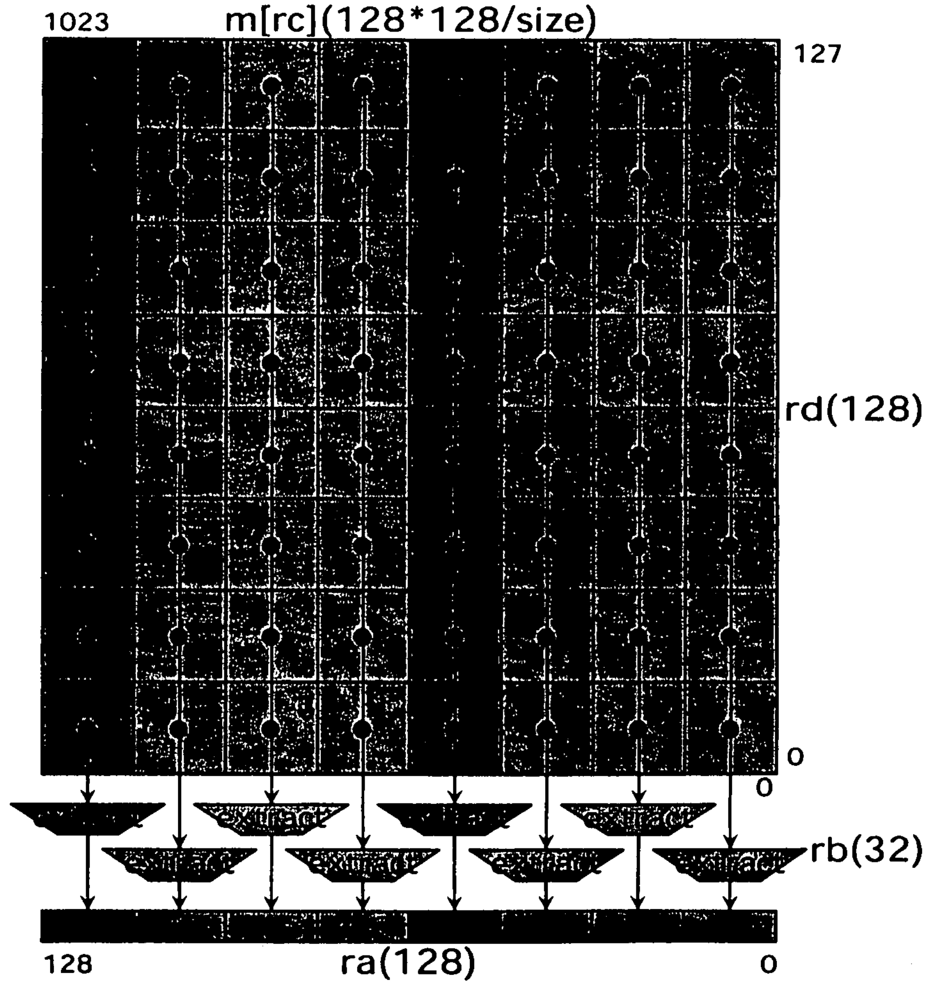
Wide multiply matrix complex

- $rd_{128} = m[rc]_{(64*64/size)} * rb_{128}$



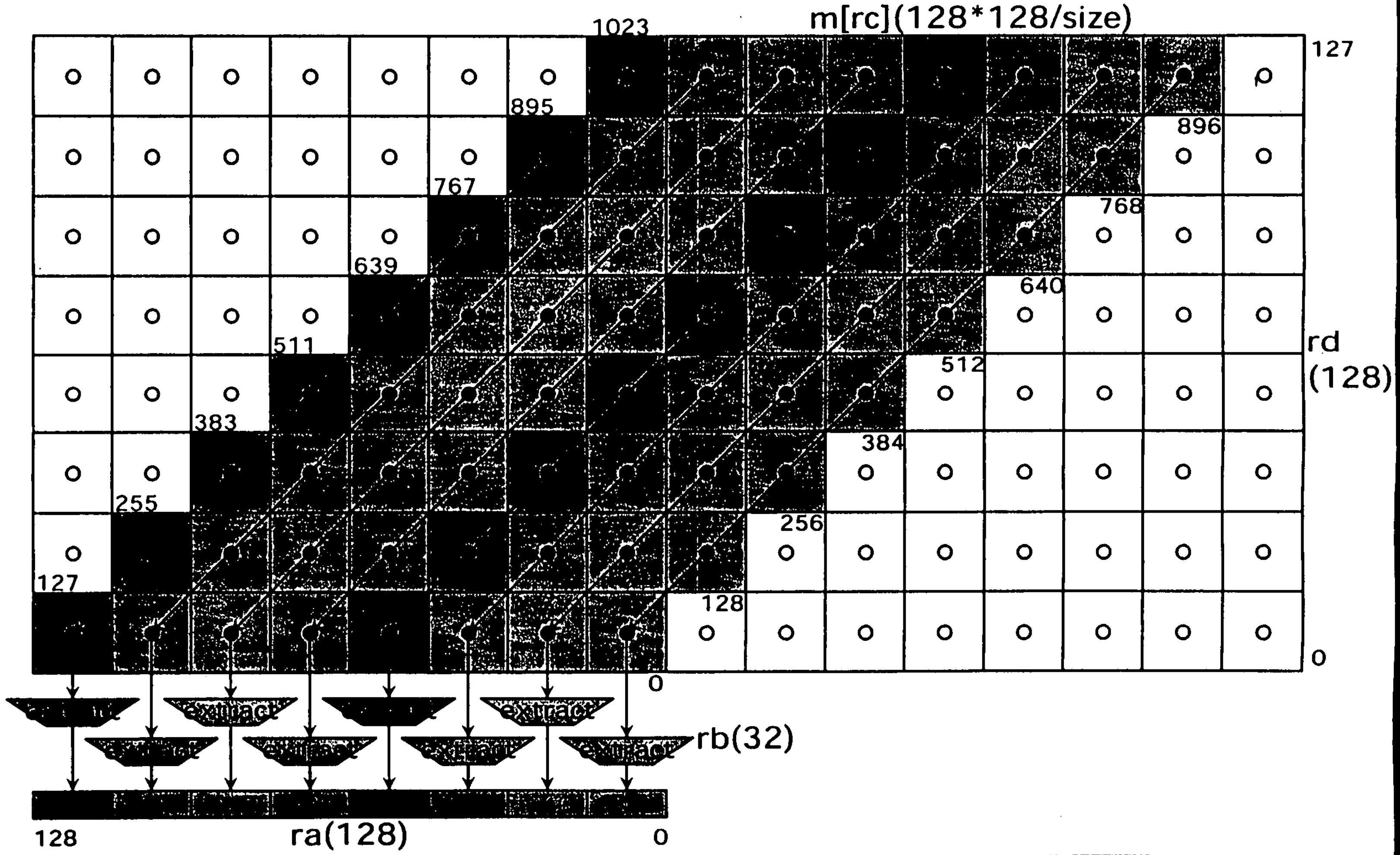
Wide multiply matrix extract

- $ra_{128} = m[rc]_{128*128/size} * rd_{128}$



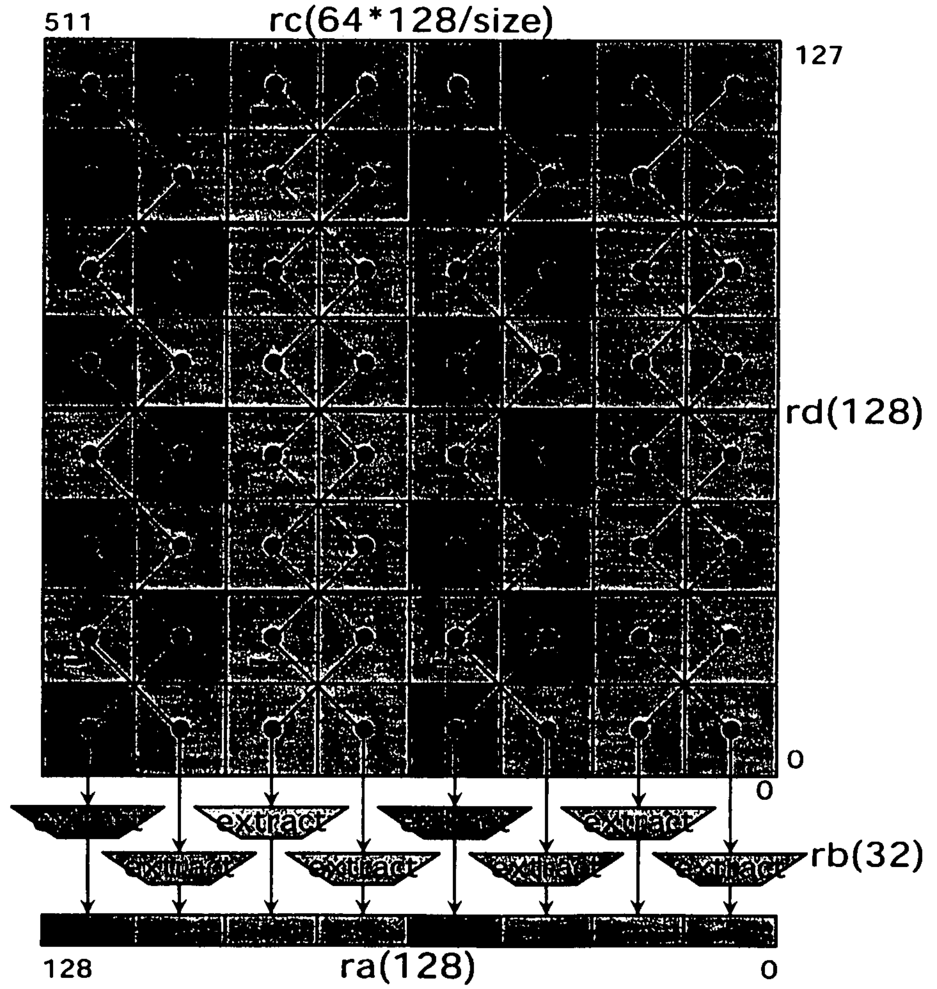
Wide multiply matrix extract

- $ra_{128} = m[rc]_{128*128/size} * rd_{128}$



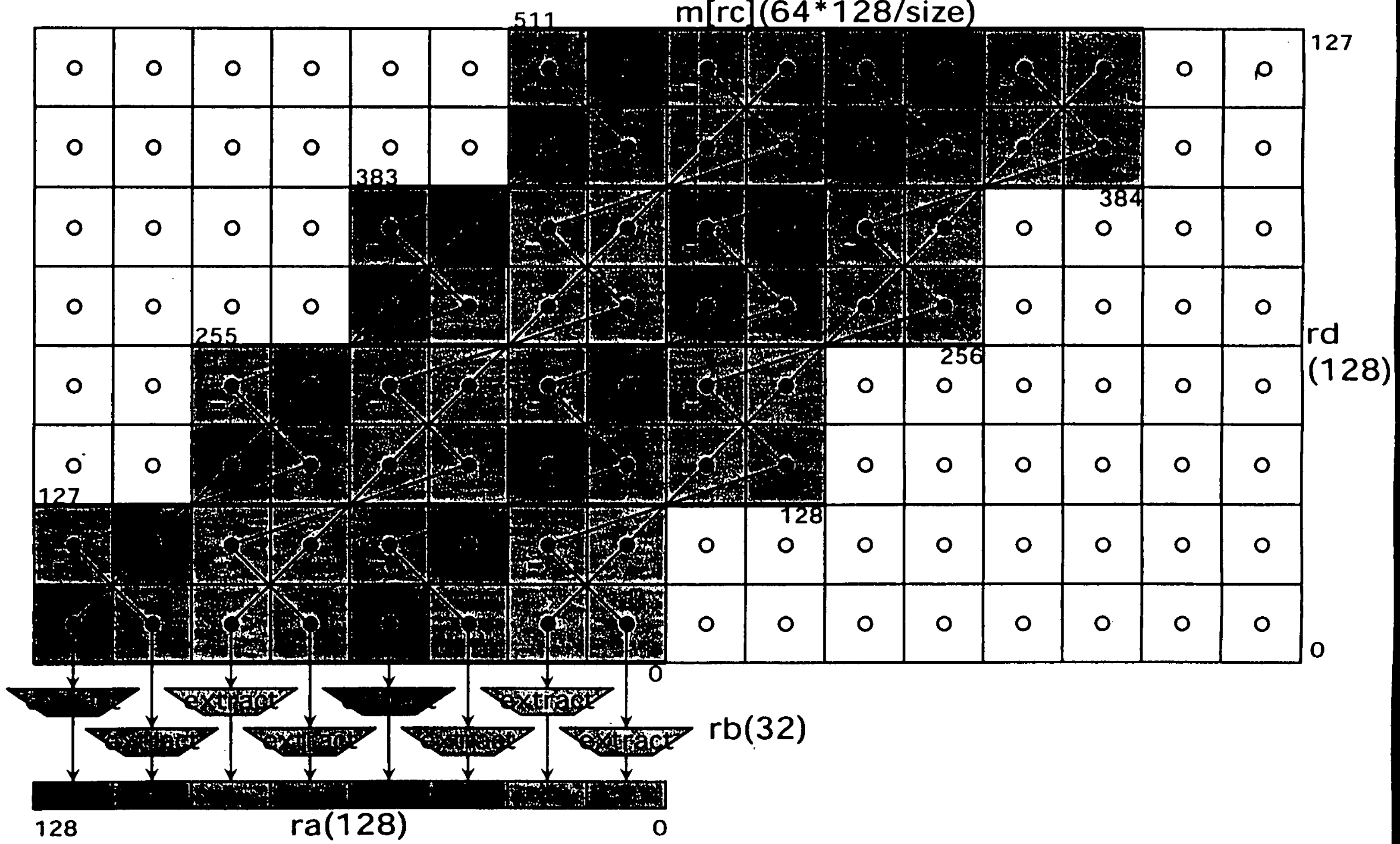
Wide multiply matrix extract complex

- $ra_{128} = m[rc]_{64*128/size} * rd_{128}$



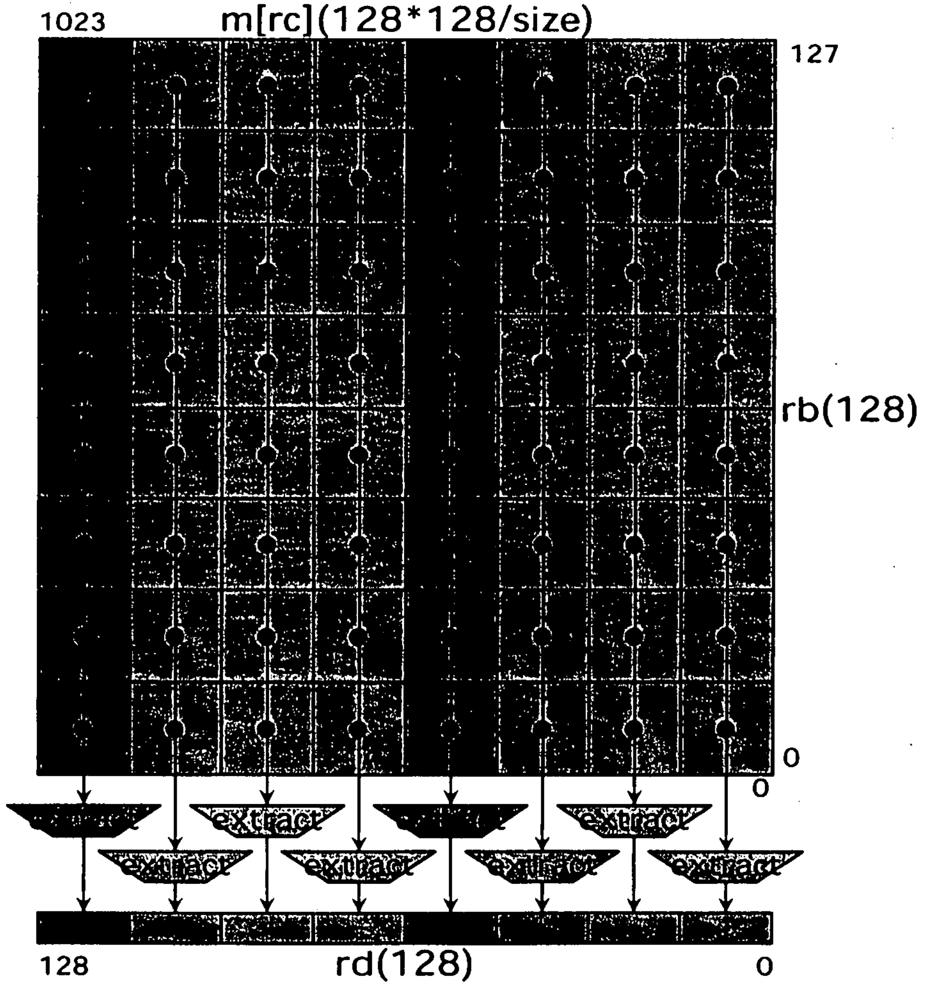
Wide multiply extract complex

- $ra_{128} = m[rc]_{64*128/size} * rd_{128}$



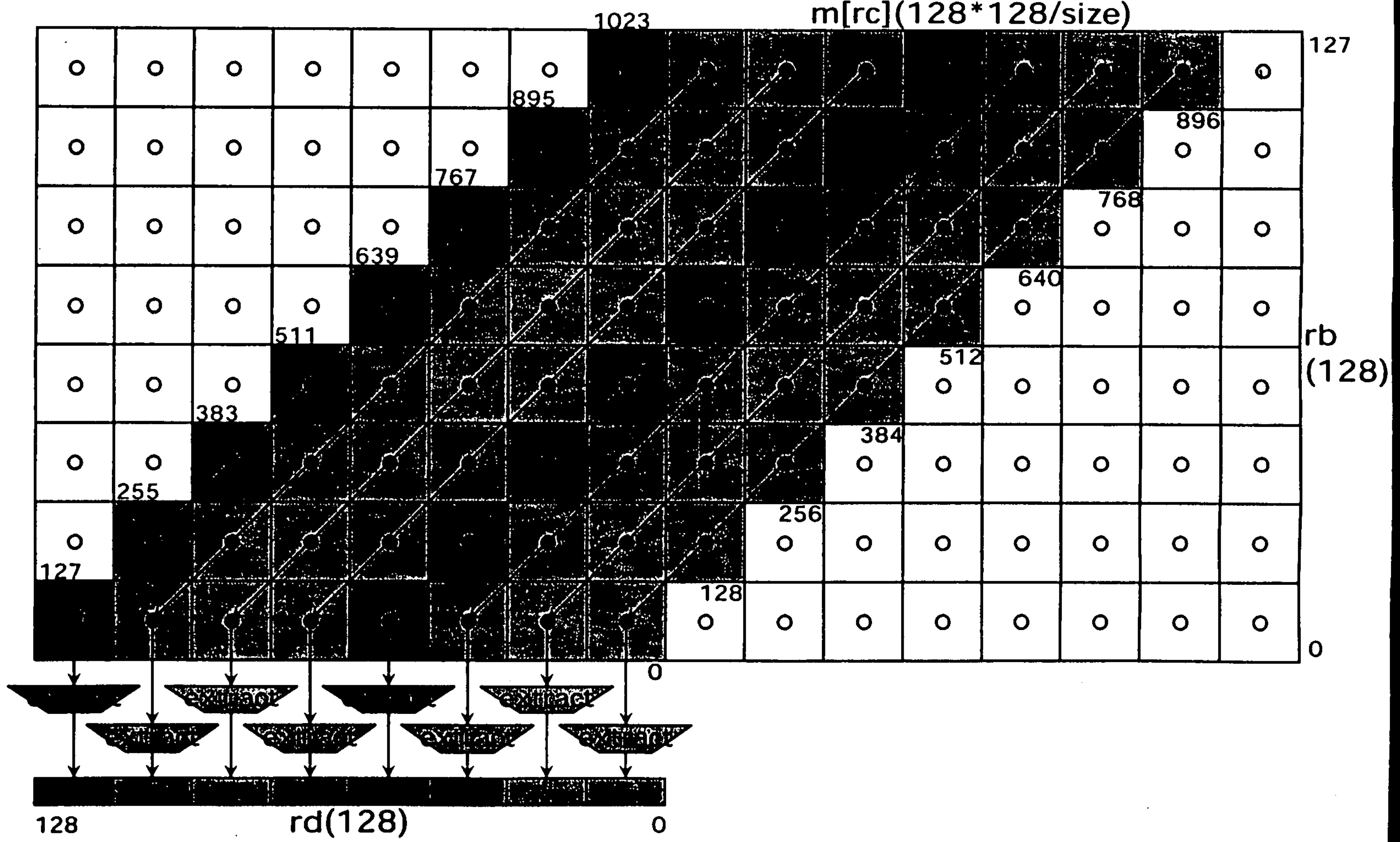
Wide multiply matrix extract immediate

- $rd_{128} = m[rc]_{128*128/size} * rb_{128}$



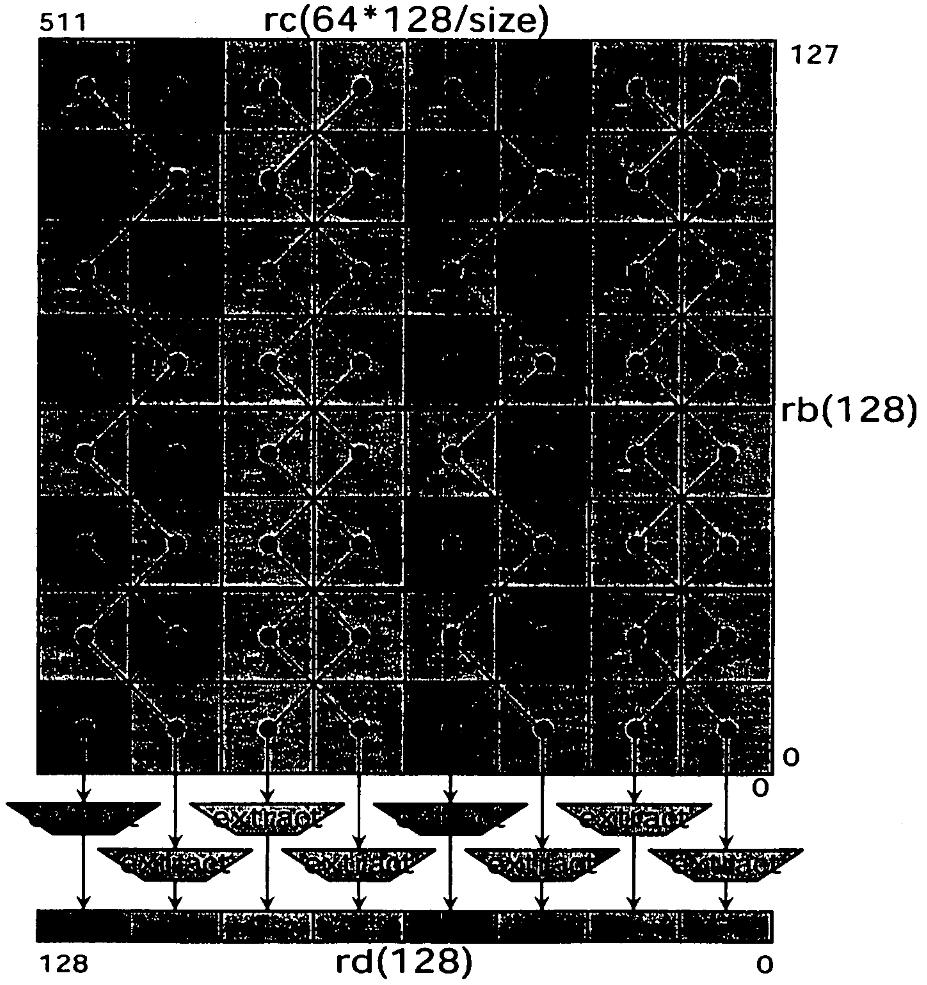
Wide multiply matrix extract immediate

- $rd_{128} = m[rc]_{128*128/size} * rb_{128}$



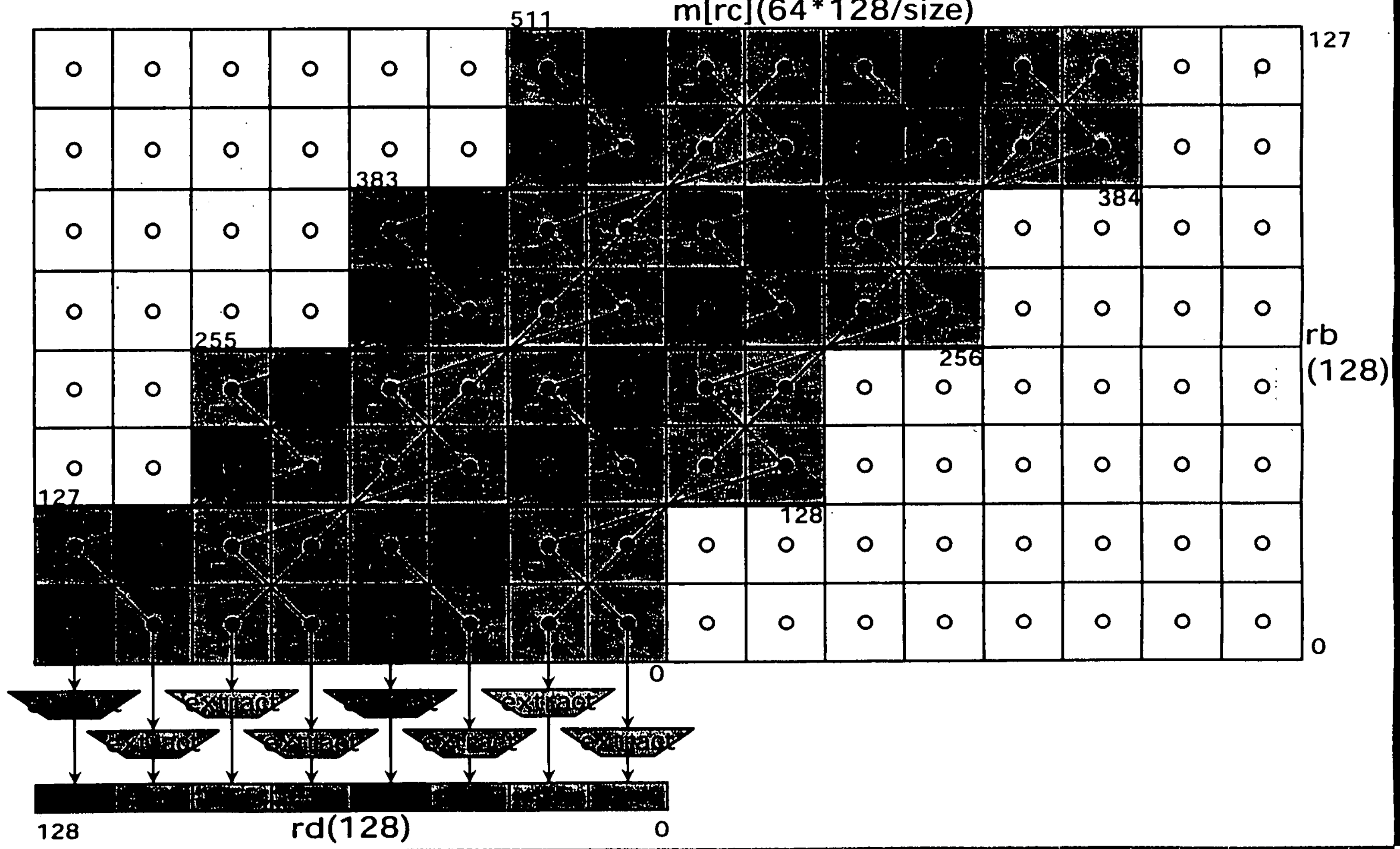
Wide multiply matrix extract immediate complex

- $rc_{128} = m[rc]_{64*128/size} * rb_{128}$



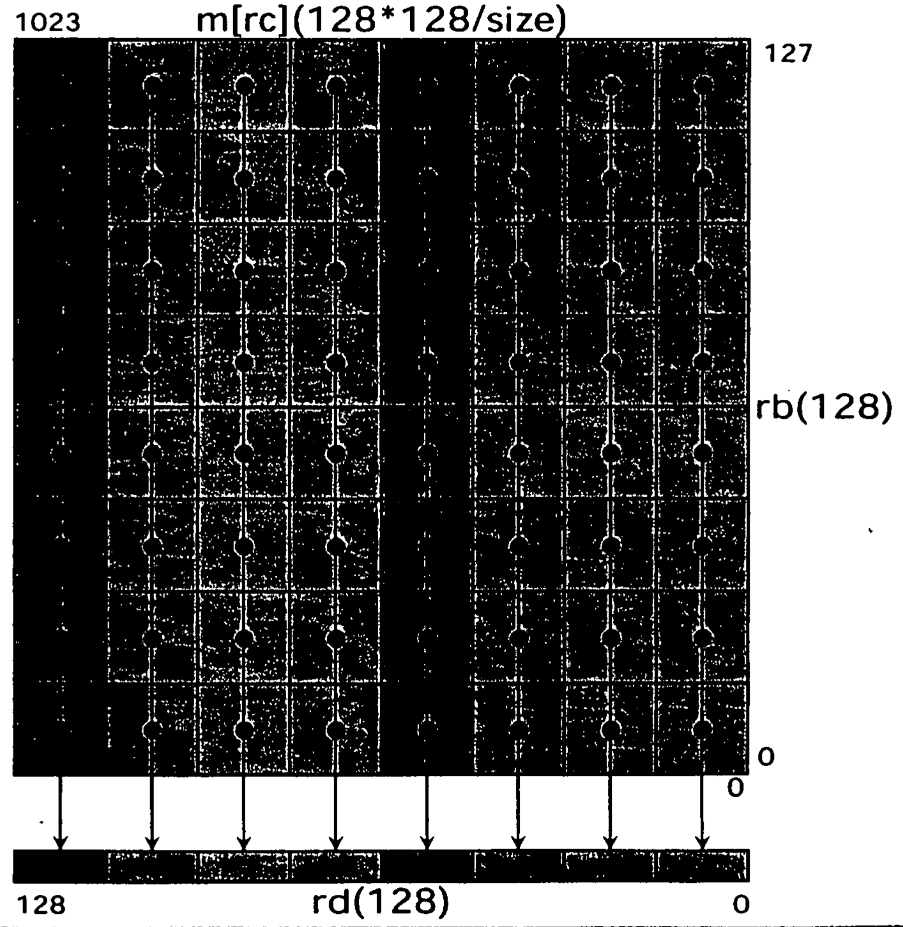
Wide multiply extract immediate complex

- $rd_{128} = m[rc]_{64*128/size} * rb_{128}$



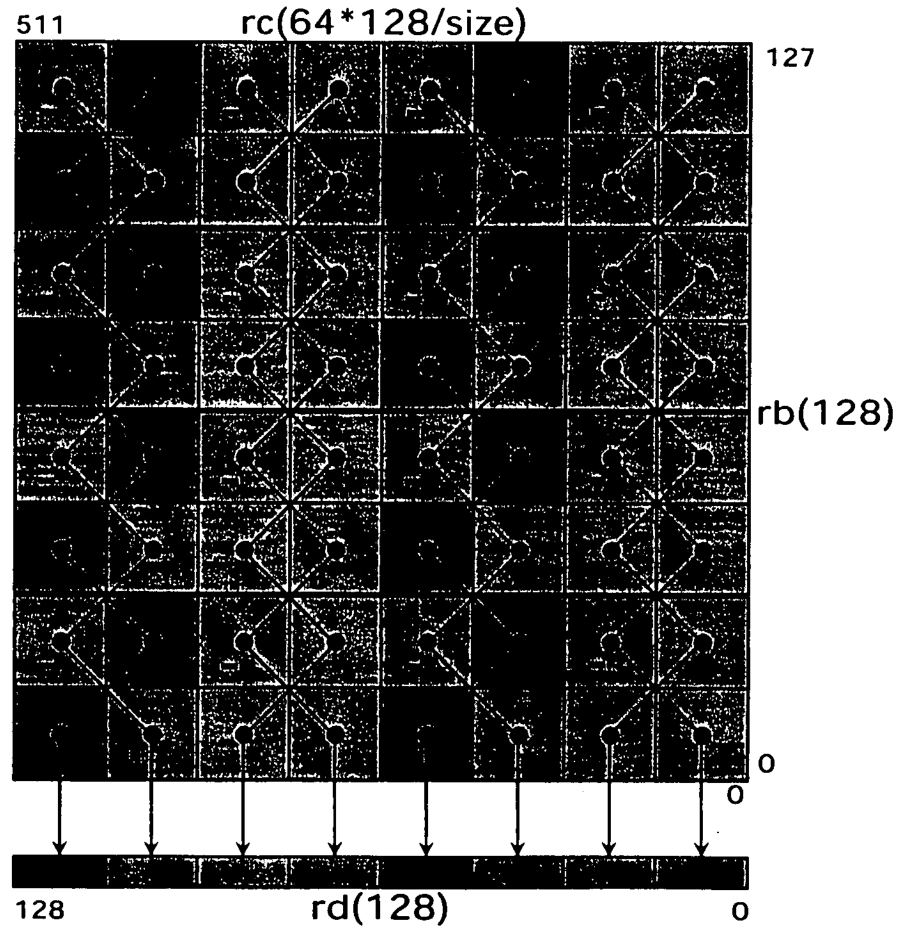
Wide multiply matrix floating-point

- $rd_{128} = m[rc]_{128*128/size} * rb_{128}$



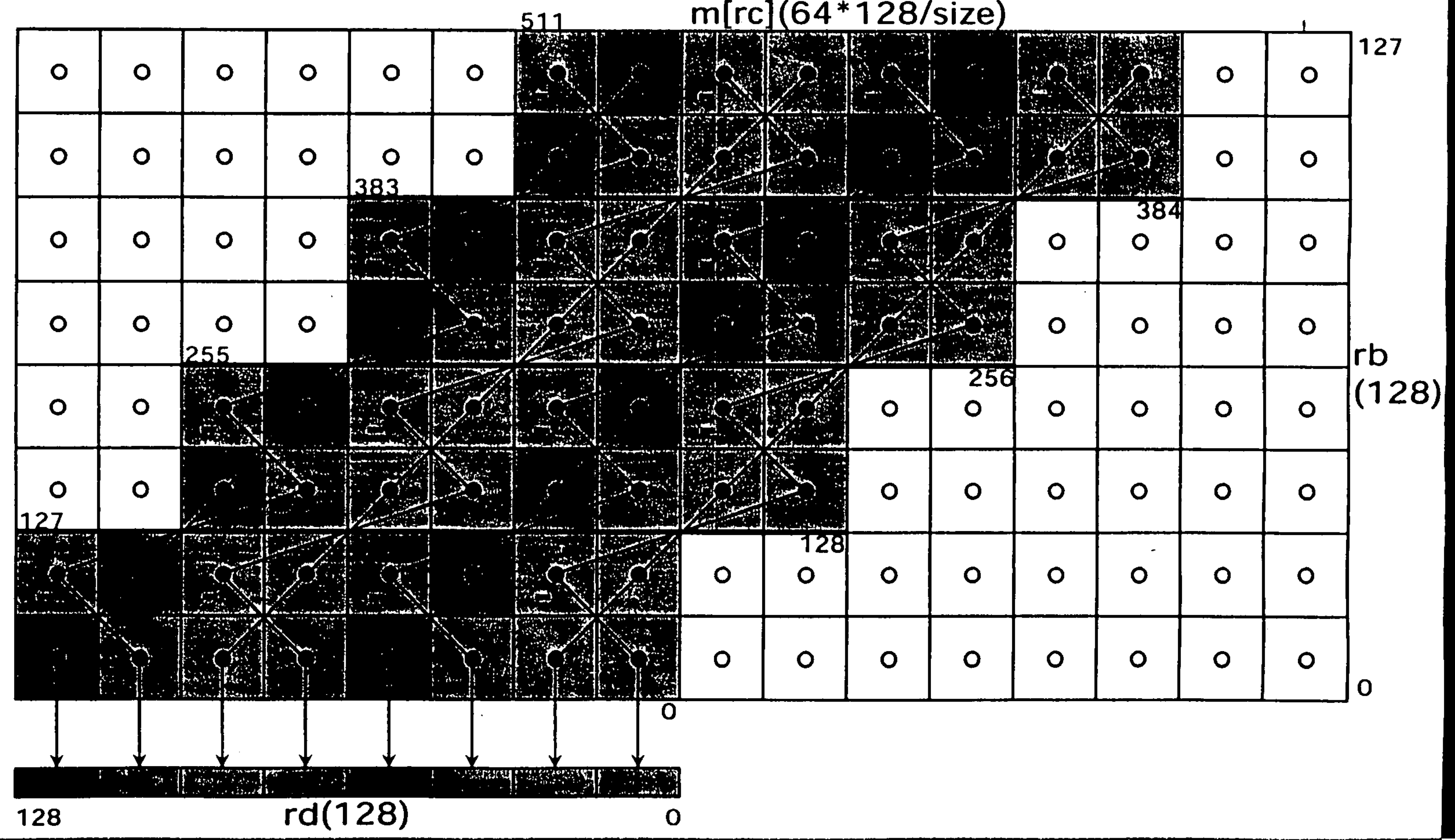
Wide multiply matrix c●mplex floating-point

- $rd_{128} = m[rc]_{64*128/size} * rb_{128}$



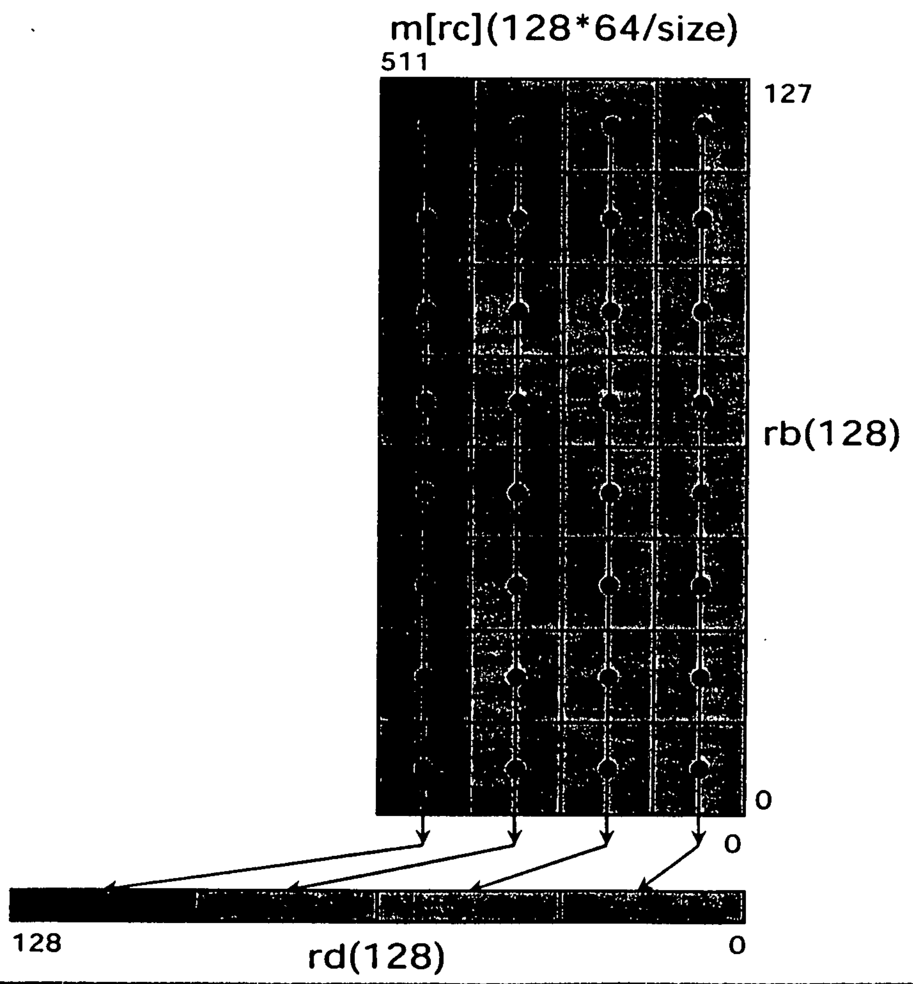
Wide multiply matrix complex floating-point

- $rd_{128} = m[rc]_{64*128/size} * rb_{128}$



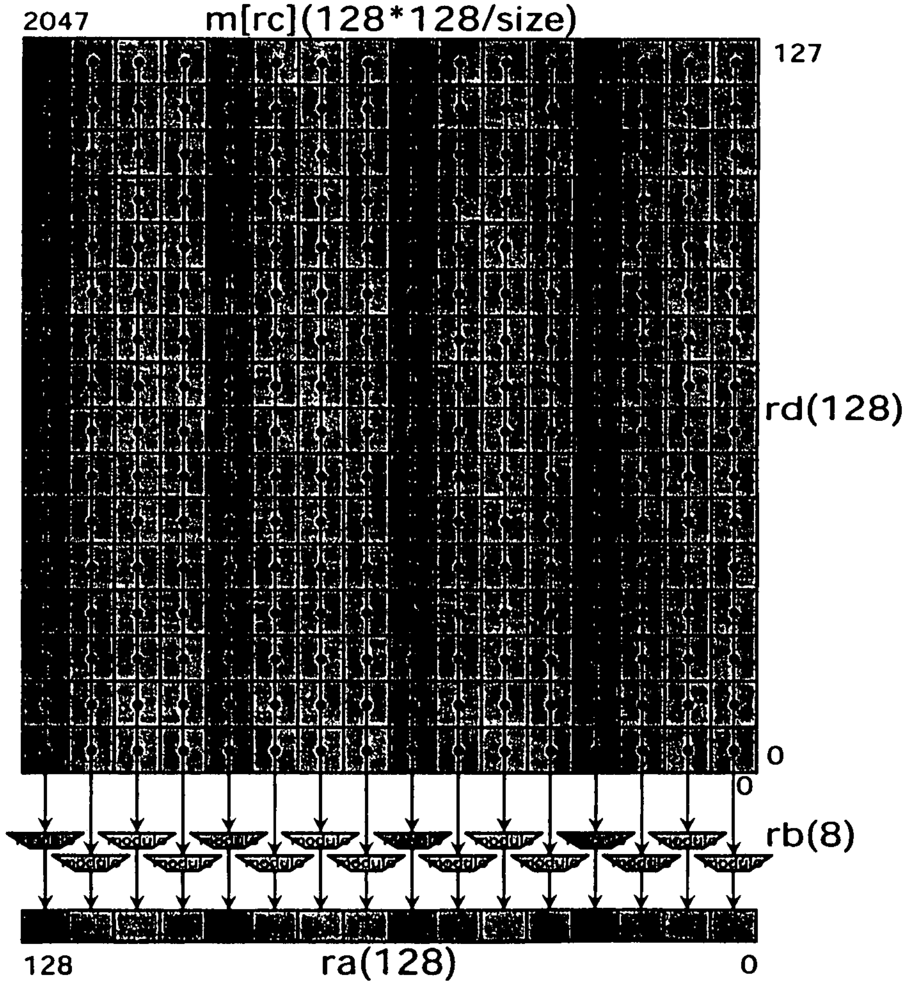
Wide multiply matrix polynomial

- $rd_{128} = m[rc]_{(128*64/size)} * rb_{128}$



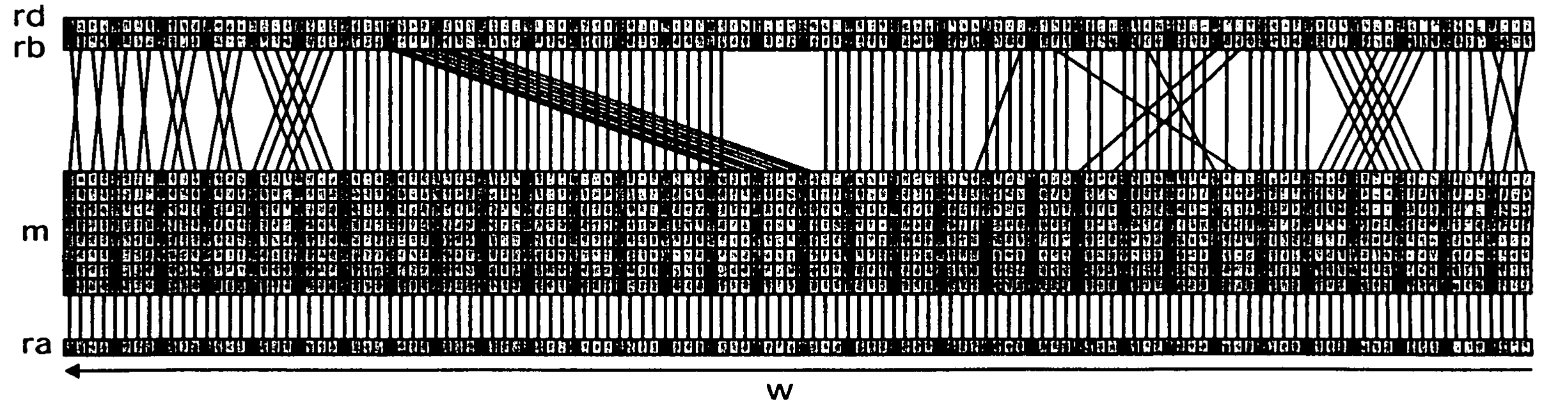
Wide multiply matrix Galois

- $ra_{128} = m[rc]_{128*128/size} * rd_{128} \bmod rb_8$



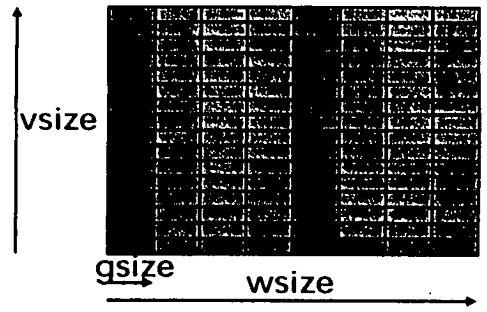
Wide switch

- $j(i) = m[rc]_{7w+i,6w+i,5w+i,4w+i,3w+i,2w+i,w+i,i}$
- $ra_i = (rd||rb)_j$, $i=0..127$
- rc specifies address and w
  - ◆ rc = base + w/2



Wide Table

- Table lookup
  - msize: total table size
  - wsize: table width
  - vsize: table depth
  - gsize: Group size (table granularity)
- $j(i) = b_{\text{vsize}-1+i..i*\text{wsize}+i \mid \text{wsize}-1..0}$
- $rd_{i+\text{gsize}-1..i} = m[rc]_{j+\text{gsize}-1..j}$, $i=0..128\text{-gsize by gsize}$
- rc specifies address, msize, wsize
  - rc = base + msize/16 + wsize/16
  - vsize = msize/wsize



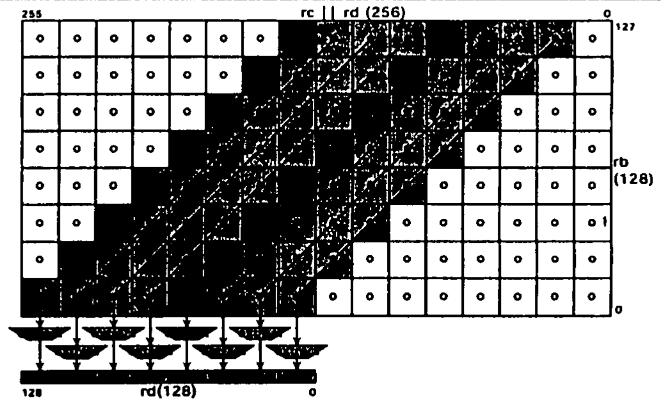
MicroUnity

Summary

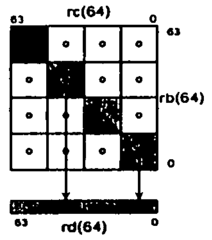
- Order-of-magnitude multiply performance increase
  - matrix multiply
  - convolve
- Wide switch: bit permutation
- Wide select: table lookup

BroadMX™ vs. MMX™

- Convolve Extract
  - 64 Multiplies
  - 56 Adds
  - 8 Extract w/round
- MMX Instructions
  - 16 MOV
  - 16 PMADDWD
  - 12 PADDD
  - 8 PSHW
  - 4 PSHR
  - 2 PACK
  - 58 total

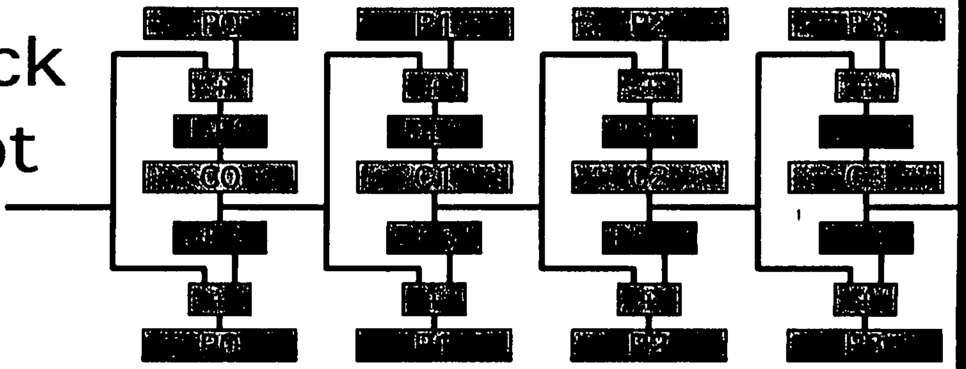


16

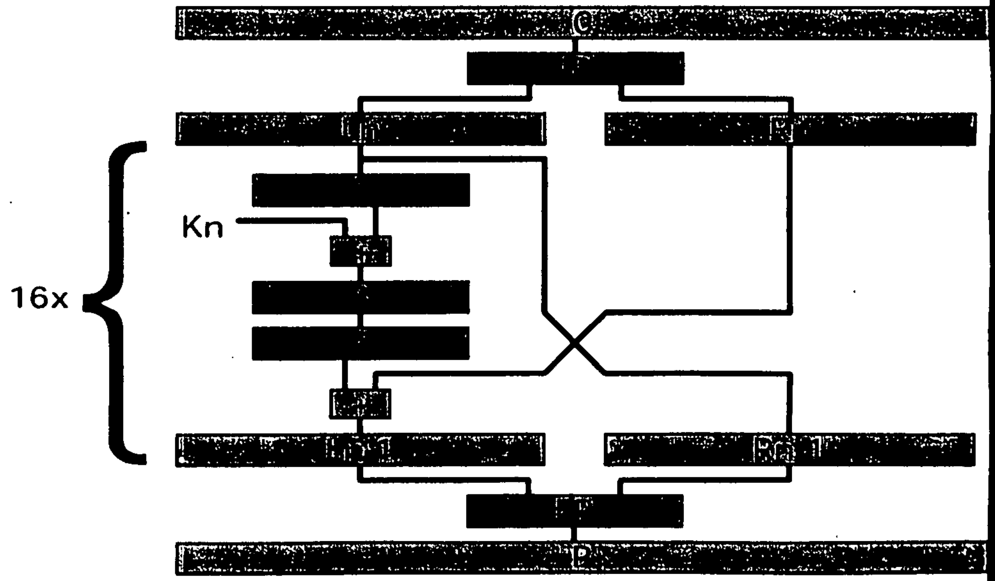


DES decryption

- CBC (Cypher Block Chaining) decrypt uses parallelism between blocks



- DES decrypt
  - E expansion
  - + key xor
  - S substitution
  - P permutation
  - + data xor

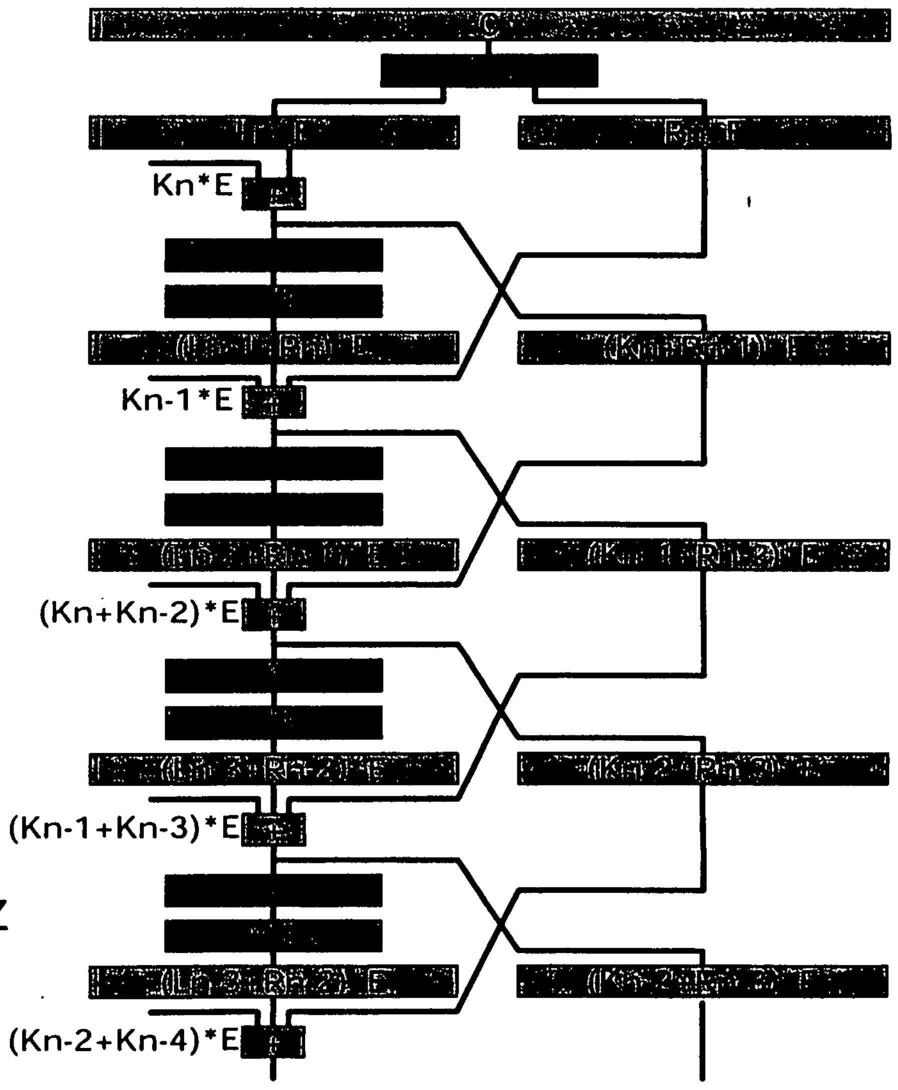


Software DES

- Optimizations
  - 2 blocks/register
  - 4 blocks at once
  - distribute E
  - combine + +
- Code

```
K,+  L.128, G.XXX
S    W.TRANSLATE
PE   W.SWITCH
```

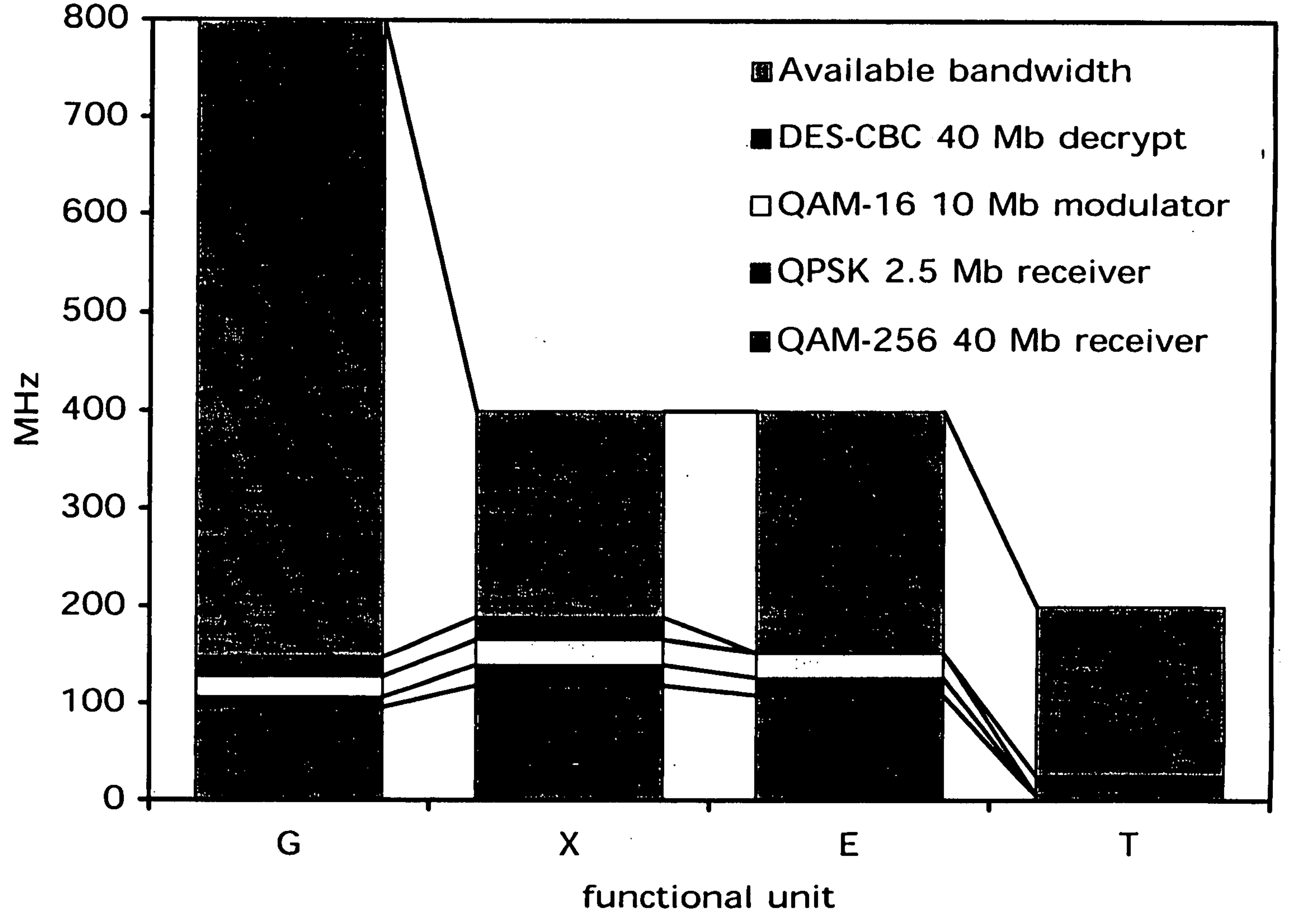
- Performance
  - 52 cycles/4 blocks
  - 985Mbps@200MHz
  - 10x per clock over fastest sw DES



Software DES

- DES standard at end of 20 year life
  - brute-force code-breaking
    - $10000 RSA DES Challenge
    - Electronic Frontier Foundation (EFF)
      - 56 hours to crack
      - $200k to design and build
  - FIPS standard expire this year
- Handles DES extensions
  - larger keys, bigger S-boxes
  - more rounds, larger blocks
  - soft S-boxes and P-boxes
- AES standard in development
  - 15 official candidates
  - new standard unpredictable

Instruction bandwidth for cable modem



MicroUnity

Software tools

- Compiler-based development tools
  - C, C++ compiler
    - intrinsic functions, function inlining
    - register allocation, code scheduling
    - future: automatic parallelisation
  - object-module tools
    - linker, libraries, debugger
- OS: RT microkernel, Linux
- DSP libraries
- Sophisticated tools
    - Mathematica: symbolic verification
    - GOPS: cross-development library

MicroUnity

Still to come

- Key code examples
  - signal
  - graphics
  - channel
- Architectural review
- Microarchitectural features
- Wide architecture